Spis treści

1. Opis systemu……...4
1.1 Role w systemie……..4
1.2 Opis funkcji systemu……..4
1.3 Funkcje systemowe przypisane do użytkowników……....5
1.4 Use Case Diagrams……....7

2. Schemat Bazy Danych……9
2.1 Schemat……..9

3. Tabele……...10
3.1 Company……....10
3.2 Days……10
3.3 Conference……10
3.4 Conference Reservation……11
3.5 DayReservation……11
3.6 Participant……12
3.7 ParticipantsForDay……...12
3.8 Payment ……...12
3.9 PriceTreshold……13
3.10 Workshops……..13
3.11 WorkshopsParticipants……....14
3.12 WorkshopsReservation ……...14
3.13 Usuwanie Tabel……15

4. Widoki……15
4.1 UnpaidReservations……15
4.2 ConferenceDayParticipants……..16
4.3 ConferenceWorkshopParticipants……..16
4.4 MostActiveCustomers……....16
4.5 CustomersWithNotFilledReservation……17
4.6 FreePlacesPerConferenceDay……17
4.7 FreePlacesPerConferenceWorkshop……..17
4.8 ConferencePopularity……17
4.9 ConferenceIncome……18
4.10 WorkshopsPopularity……18
4.11 MostActiveParticipants……...18
4.12 PaidReservation……19
4.13 AmountToPayForConferenceReservation……...19
4.14 OneParticipantCcompanies……..19
4.15 ManyParticipantsCompanies……....19
4.16 ParticipantsWithCompanies……....20

4.17 StudentsParticipants……..……...20
4.18 NotStudentsParticipants…….……..20

5. Procedury…….....20
5.1 AddConference…….……...……...20
5.2 UpdateConferenceDayCapacity…….……21
5.3 AddWorkshop…….…...…...…...…...22
5.4 RemoveConference…….…...…...……23
5.5 RemoveWorkshop…….…...…...…….....23
5.6 AddPriceThreshold…….…...…...…….24
5.7 RemovePriceThreshold…….……...……..24
5.8 UpdateWorkshopCapacity…….……...……..25
5.9 AddCompany…….…...…...…….....26
5.10 BookConference…….…...…...…....……26
5.11 BookDay…….…...…...…...…...……..28
5.12 BookWorkshop…….…...…...…...…....29
5.13 CancelConferenceReservation……...…..30
5.14 CancelDayReservation…….…...…...…...31
5.15 CancelWorkshopReservation……...……31
5.16 CancelUnpaidOnTimeReservation…….....32
5.17 AddPayment…….…...…...…….....33
5.18 AddParticipantToDay…….…...…….....33
5.19 RemoveParticipantFromDay…….…….....34
5.20 AddParticipantToWorkshop…….…….....35
5.21 RemoveParticipantFromWorkshop……...…...36
5.22 AddParticipant…….…...…...…….....36
5.23 AddDay…….…...…...…...…….....37
5.24 RemoveDay…….…...…...……...38

6. Funkcje…….....38
6.1 CalculateAmountToPay…….…...……38
6.2 DayFreePlaces…….…...…...…...……..39
6.3 WorkshopFreePlaces…….…....…....……...40
6.4 ConferenceDays…….…...…...…...……...40
6.5 WorkshopCollision…….…...…...……..40
6.6 ConferenceFreePlaces…….…...…...……41
6.7 ConferenceAttendance…….……..41
6.8 DayParticipantsList…….……...41
6.9 WorkshopParticipantsList…….……42
6.10 ParticipantWorkshopsList…….……...42
6.11 CountStudentsNumber…….…….....42
6.12 DayWorkshops…….……...42
6.13 ParticipantDayList…….……...43

7. Triggery……...43
7.1 CorrectDayDate…….…….43
7.2 CorrectDayBooking…….……...43
7.3 CorrectWorkshopBooking…….……44
7.4 NoPlaceForDay…….…….44
7.5 NoPlaceForWorkshop…….……...45
7.6 LessPlaceInWorkshopThanDay…….45
7.7 NoPlaceForParticipantInDay…….…….46
7.8 NoPlaceForParticipantInWorkshop……....46
7.9 CorrectDayCapacity…….……..47
7.10 CorrectWorkshopCapacity…….……..47
7.11 CheckWorkshopCollision…….……...48
7.12 CanStudentBeAdded…….……...48
7.13 CorrectPriceThresholdDate…….…….49
7.14 CorrectPartcipantCompany…….…….49
7.15 CorrectWorkshopParticipantDay…….50
7.16 NoMoreDayBookingAfterPayment……..50
7.17 NoMoreWorkshopBookingAfterPayment……....51

8. Generator Danych……...……...51
8.1 Opis…….……..……...51
8.2 Kod generatora…….……..51

1.Opis Systemu

1. **Role w systemie:**

    - **Administrator -** Człowiek opiekujący się całym systemem, posiada bezpośredni dostęp do bazy.
    - **Klient -** osoba, która rezerwuje miejsca w konferencji, płaci za udział, może być firmą lub osobą prywatną. Przesyła listę uczestników konferencji.
    - **Moderator/ Pracownik-**osoba, która organizuje zapisy na konferencję, jest odpowiedzialna za otrzymanie listy uczestników od klienta, a także odpowiada za kontakt z klientem. Może również na przykład dodawać nowe warsztaty do konferencji.
    - **Uczestnik-** osoba którą do udziału w konferencji zgłasza klient. Uczestniczy w konferencjach, dniach, warsztatach.o

2. **Opis funkcji systemu**

KONFERENCJE

Organizator będzie mógł tworzyć konferencje jedno lub kilkudniowe, i dodawać do nich dni, na które zapisywać się mogą klienci - indywidualni lub firmy. Klienci będą mogli przeglądać dostępne konferencje. System będzie pozwalał rezerwować miejsca klientom, a następnie dodawać na te miejsca konkretne osoby, w terminie do dwóch tygodni do konferencji. Również w tym terminie będzie mógł anulować rezerwację. Klient będzie mógł również wyświetlić listę dni konferencji i określić jaką osobę wysyła na jaki dzień konferencji. Pracownicy firmy, będą mogli przeglądać listę rezerwacji jak i również uczestników przypisanych do konkretnych dni. Mają możliwość wyświetlenia również listy firm które są spóźnione z wysłaniem listy uczestników.

WARSZTATY

Pracownicy firmy dodają warsztaty do dni konferencji, określając czy są płatne, oraz ilość miejsc. Warsztat musi być spójny, nie może mieć przerw, może być kilka w tym samym czasie. Klienci mogą się rejestrować tylko na jeden w określony czasie w okresie do 7 dni do rozpoczęcia konferencji, do tego czasu też mogą zmieniać rezerwacje.

UCZESTNICY

Uczestnicy mają dostęp jedynie do zobaczenia na jakie warsztaty i dni konferencji są zapisani. Nie mogą ani zmieniać rezerwacji ani dodawać uczestników.

OPŁATY

Klienci mają czas na opłacenie rezerwacji do tygodnia od jej złożenia. Jeżeli tego nie zrobią to rezerwacja jest anulowana. Do ceny wliczają się również ceny płatnych warsztatów. Jeśli, któryś z uczestników jest studentem, należy podać nr. legitymacji studenckiej by otrzymać 40% zniżki. Zniżkę otrzymają również jeżeli, dokonają płatności odpowiednio wcześniej, zgodnie z progami cenowymi ustalonymi przez pracowników i organizatorów. Pracownicy firmy mogą wyświetlać listę klientów którzy dokonali płatności, jak także tych którzy jej nie dokonali.

RAPORTY

Pracownicy firmy, będą mogli wyświetlać informacje takie jak lista najbardziej aktywnych klientów, a także listy uczestników do konkretnych warsztatów, co na pewno pomoże przy robieniu statystyk.

3. **Funkcje systemowe przypisane do użytkowników**

Moderator(Pracownik firmy) lub Administartor

- **PROCEDURY:**
  - AddConference
  - SetConferenceDayCapacity
  - AddWorkshop
  - AddCompany
  - RemoveConference
  - RemoveConferenceDay
  - RemoveWorkshop
  - AddPriceThreshold
  - RemovePriceThreshold
  - UpdateWorkshopCapacity
  - CancelUnpaidOnTimeReservations

- **WIDOKI:**
  - UnpaidReservations
  - ConferenceDayParticipants
  - ConferenceWorkshopParticipants
  - MostActiveCustomers
  - CustomersWithNotFilledReservation
  - FreePlacesPerConferenceDay
  - FreePlacesPerConferenceWorkshop
  - ConferencePopularity
  - ConferenceIncome
  - WorkshopsPopularity
  - MostActiveParticipants
  - PaidReservation
  - AmountToPayForConferenceReservation

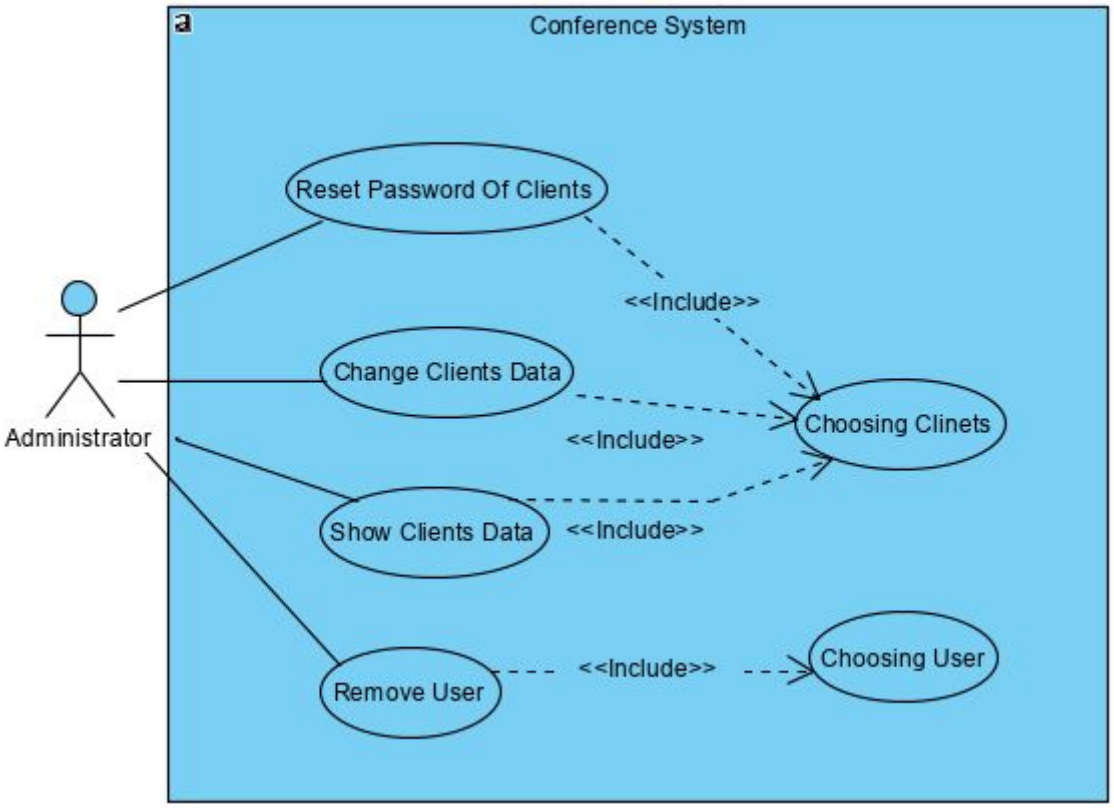
    - OneParticipantCompanies
    - ManyParticipantsCompanies
    - ParticipantsWithCompanies
    - StudentsParticipants
    - NotStudentsParticipants

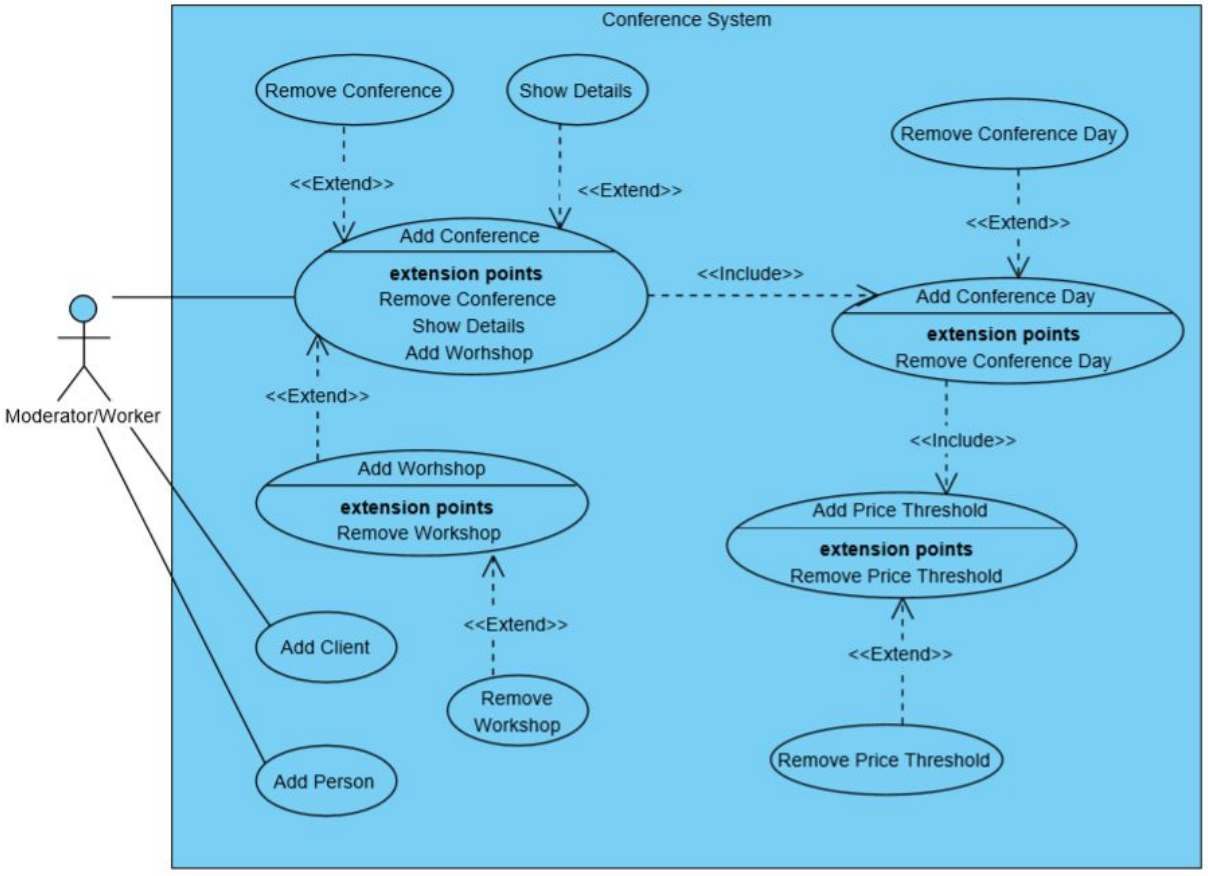
- **FUNKCJE:**
    - CalculateAmountToPay
    - DayFreePlaces
    - WorkshopFreePlaces
    - ConferenceDays
    - CheckWorkshopCollision
    - ConferenceFreePlaces
    - ConferenceAttendance
    - DayParticipantsList
    - WorkshopParticipantsList
    - ParticipantWorkshopsList
    - CountStudentsNumber
    - DayWorkshops

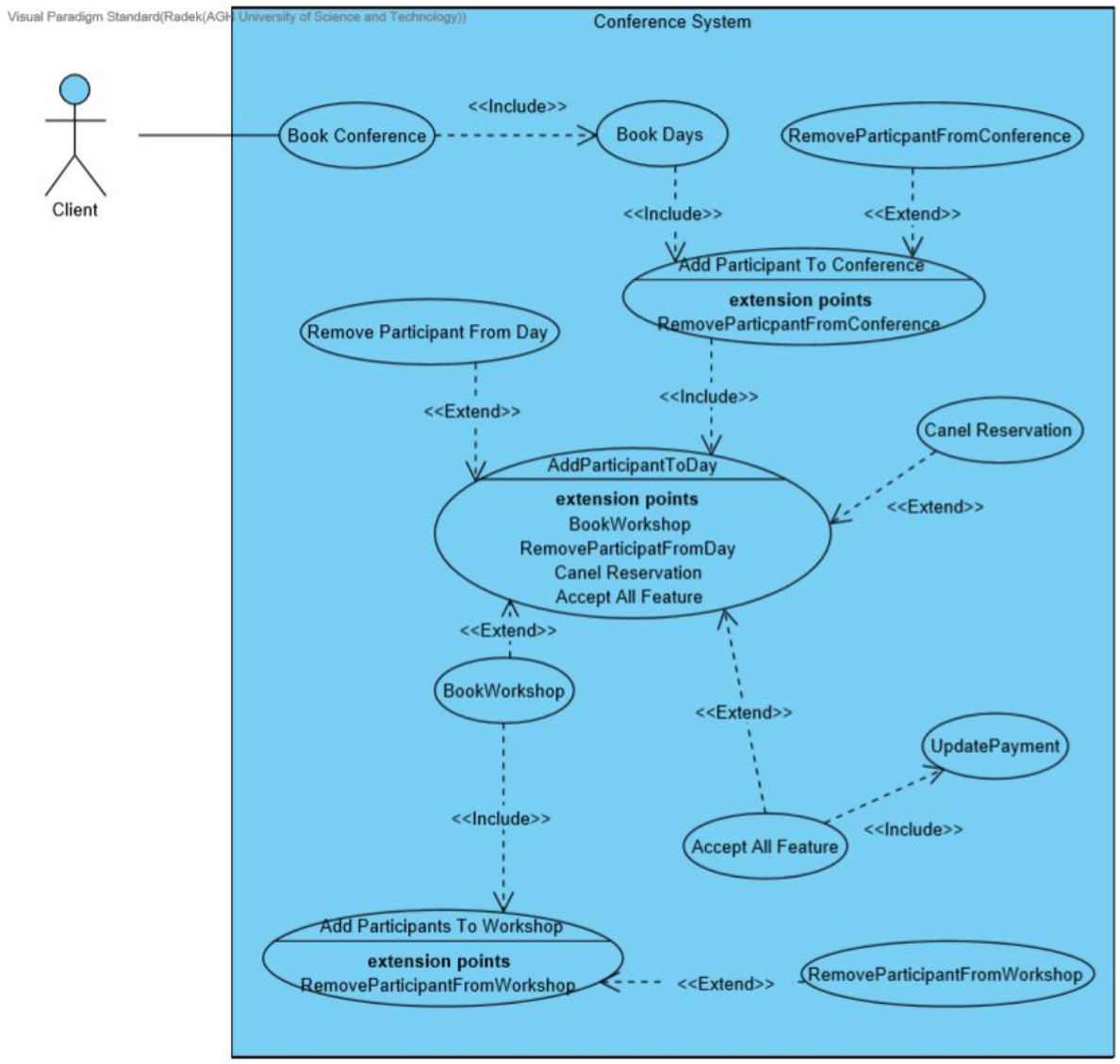
Klient:

- **PROCEDURY:**
    - AddCompany
    - BookConference
    - BookWorhshop
    - BookDay
    - AddPayment
    - CancelConferenceReservation
    - CancelDayReservation
    - CancelWorkshop
    - AddParticipantToWorkshop
    - AddParticipantToDay
    - RemoveParticipantFromDay
    - RemoveParticipantFromWorkshop
- **WIDOKI:**
    - FreePlacesPerConferenceDay
    - FreePlacesPerConferenceWorkshop
- **FUNKCJE:**
    - CalculateAmountToPay
    - ConferenceFreePlaces
    - CheckWorkshopCollision
    - ConferenceDays
    - WorkshopFreePlaces
    - DayFreePlaces
    - ParticipantWorkshopsList
    - DayWorkshops

4. Diagramy Use Case

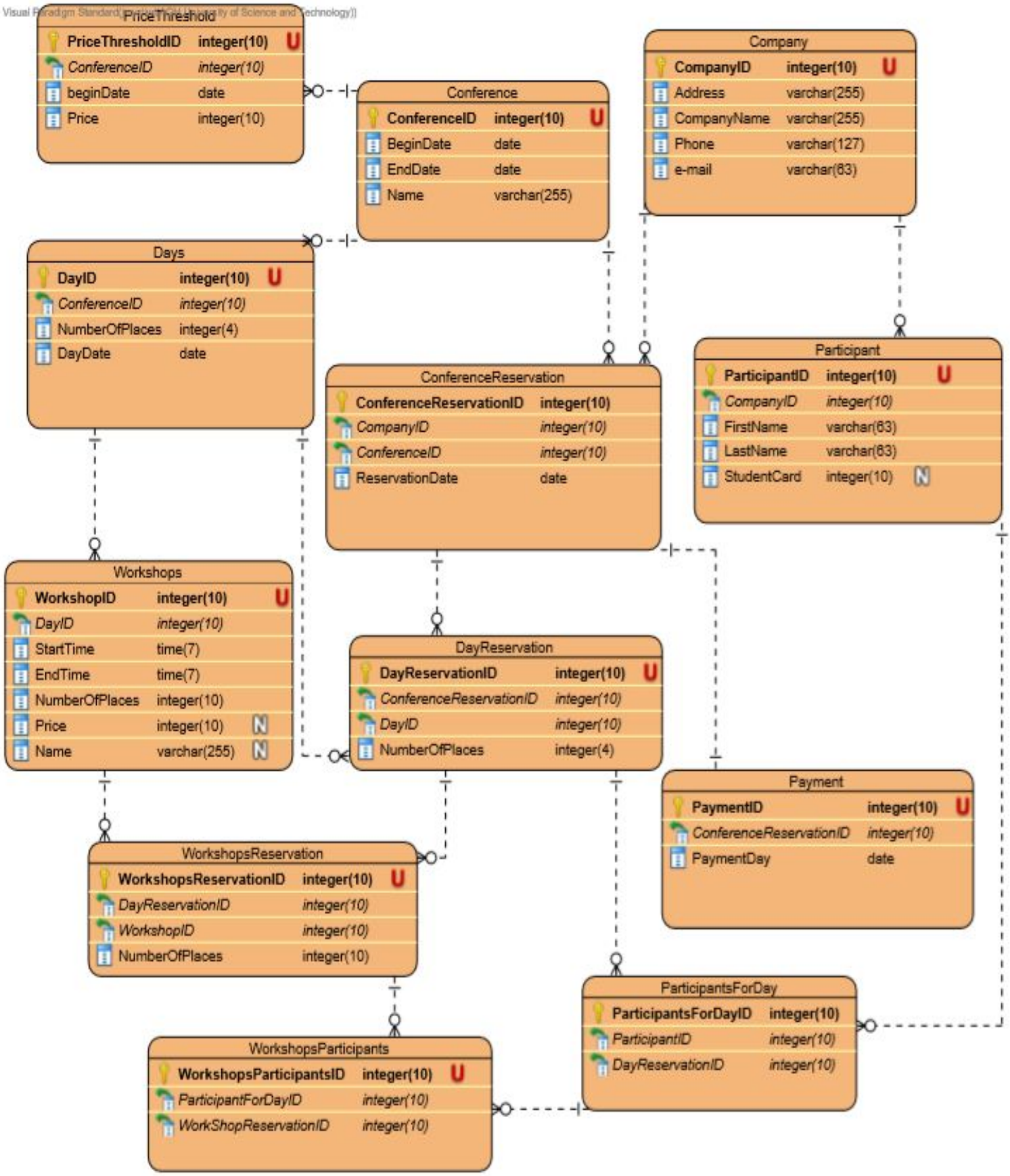





Visual Paradigm Standard(Radek(AGH University of Science and Technology))
Conference System
Client
Book Conference
<<Include>>
Book Days
RemoveParticpantFromConference
<<Include>>
<<Extend>>
Add Participant To Conference
extension points
RemoveParticpantFromConference
Remove Participant From Day
<<Extend>>
<<Include>>
Canel Reservation
AddParticipantToDay
extension points
BookWorkshop
RemoveParticipatFromDay
Canel Reservation
Accept All Feature
<<Extend>>
<<Extend>>
BookWorkshop
<<Extend>>
UpdatePayment
<<Include>>
Accept All Feature
<<Include>>
Add Participants To Workshop
extension points
RemoveParticipantFromWorkshop
<<Extend>>
RemoveParticipantFromWorkshop

2. Schemat Bazy Danych

2.1 Schemat



3. Tabele

3.1 Company

Przechowuje informacje o firmach, klientach.
CompanyID - Identyfikator firmy/ klienta
Adress - Adres firmy / klienta
CompanyName - Nazwa firmy/ klienta
Phone - telefon kontaktowy do klienta
[e-mail] - adres mailowy do klienta

```
CREATE TABLE Company (CompanyID int IDENTITY(1,1) NOT NULL , Address varchar(255) NOT NULL,
CompanyName varchar(255) NOT NULL, Phone varchar(127) NOT NULL, [e-mail] varchar(63) CHECK
([e-mail] LIKE '_%@_%._%')NOT NULL, PRIMARY KEY (CompanyID) );
```

3.2 Days

Tabela przechowuje informacje o dniach w których dana konferencja się odbywa.
DayID - identyfikator danego dnia
ConferenceID - odniesienie do danej konferencji, podczas której jest dany dzień
NumberOfPlaces - Ilosć miejsc na danym dniu konferencji
DayDate - data dnia w którym odbywa się konferencja

```
CREATE TABLE Days (DayID int IDENTITY(1,1) , ConferenceID int NOT NULL,
NumberOfPlaces int CHECK(NumberOfPlaces > 0)  NOT NULL default 0, DayDate date NOT NULL,
PRIMARY KEY (DayID), UNIQUE (ConferenceID, DayDate));
```

```
ALTER TABLE Days ADD CONSTRAINT FKDays575639 FOREIGN KEY (ConferenceID) REFERENCES
Conference (ConferenceID) on delete cascade;
```

3.3 Conference

Tabela pzechowuje dane o Konferencji

ConferenceID - Identyfikator
iBeginDate - data rozpoczęcia konferencji
EndDate - data zakończenia konferencji
Name - Nazwa Konferencji

```
CREATE TABLE Conference (ConferenceID int IDENTITY(1,1) NOT NULL, BeginDate date NOT NULL,
EndDate date NOT NULL, Name varchar(255) NOT NULL, PRIMARY KEY (ConferenceID), CONSTRAINT
CHK_DATE CHECK (DATEDIFF(day,BeginDate,EndDate) >= 0));
```

3.4 Conference Reservation

Tabela przechowuje informacje o rezerwacjach konferencji, wykonanych przez klienta.
ConferenceReservationID - identyfikator rezerwacji
CompanyID - identyfikator klienta, który wykonał rezerwacje
ConferenceID - identyfikator konferencji, której dotyczy rezerwacja
ReservationDate - data wykonania rezerwacji

```
CREATE TABLE ConferenceReservation (ConferenceReservationID int IDENTITY(1,1) NOT NULL,
CompanyID int NOT NULL, ConferenceID int NOT NULL, ReservationDate date NOT NULL, PRIMARY
KEY (ConferenceReservationID));
```

```
ALTER TABLE ConferenceReservation ADD CONSTRAINT FKConference392607 FOREIGN KEY
(ConferenceID) REFERENCES Conference (ConferenceID) on delete cascade;
ALTER TABLE ConferenceReservation ADD CONSTRAINT FKConference595044 FOREIGN KEY (CompanyID)
REFERENCES Company (CompanyID);
```

3.5 DayReservation

Tabela przechowuje informacje jaką ilość miejsc zarezerwował klient ,na jaki dzień konferencji.
DayReservationID - identyfikator rezerwacji danego dnia konferencji
ConferenceReservationID - identyfikator rezerwacji konferencji, dla której zarezerwowano dany dzień
DayID - identyfikato dnia, którego dotyczy dana rezerwacja
NumberOfPlaces - Liczba miejsc, którą zarezerwowano

```
CREATE TABLE DayReservation (DayReservationID int IDENTITY(1,1) NOT NULL,
ConferenceReservationID int NOT NULL, DayID int NOT NULL, NumberOfPlaces int
CHECK(NumberOfPlaces > 0) NOT NULL, PRIMARY KEY (DayReservationID), UNIQUE
(ConferenceReservationID, DayID));
```

```
ALTER TABLE DayReservation ADD CONSTRAINT FKDayReserva734297 FOREIGN KEY
(ConferenceReservationID) REFERENCES ConferenceReservation (ConferenceReservationID) ;
ALTER TABLE DayReservation ADD CONSTRAINT FKDayReserva96605 FOREIGN KEY (DayID) REFERENCES
Days (DayID) on delete cascade;
```

3.6 Participant

Tabela przechowuje informacje o uczestnikach konferencji, z danych firm.
ParticipantID - identyfikator uczestnika
CompanyID - identyfikator firmy uczestnika
FirstName - Imię uczestnika
LastName - Nazwisko uczestnika
StudentCard - Numer Legitymacji studenckiej, jeżeli jest nullem to znaczy, że użytkownik nie jest studentem

```
CREATE TABLE Participant (ParticipantID int IDENTITY(1,1) NOT NULL, CompanyID int NOT NULL,
FirstName varchar(63) NOT NULL, LastName varchar(63) NOT NULL, StudentCard int CHECK
(StudentCard >= 0 ) NULL, PRIMARY KEY (ParticipantID));
```

```
ALTER TABLE Participant ADD CONSTRAINT FKParticipan762161 FOREIGN KEY (CompanyID) REFERENCES
Company (CompanyID);
```

3.7 ParticipantsForDay

Tabela łączy uczestnika z danym dniem
ParticipantsForDayID - Identyfikator
ParticipantID - Identyfikator uczestnika
DayReservationID - identyfikator rezerwacji, na podstawie której zgłoszony jest uczestnik do udziału w danym dniu konferencji

```
CREATE TABLE ParticipantsForDay (ParticipantsForDayID int IDENTITY(1,1) NOT NULL,
ParticipantID int NOT NULL, DayReservationID int NOT NULL, PRIMARY KEY
(ParticipantsForDayID), UNIQUE (ParticipantID, DayReservationID));
```

```
ALTER TABLE ParticipantsForDay ADD CONSTRAINT FKParticipan276571 FOREIGN KEY
(DayReservationID) REFERENCES DayReservation (DayReservationID) on delete cascade;
ALTER TABLE ParticipantsForDay ADD CONSTRAINT FKParticipan37239 FOREIGN KEY (ParticipantID)
REFERENCES Participant (ParticipantID) ;
```

3.8 Payment

Tabela przechowuje informacje o czy i kiedy została wykonana płatność za rezerwację.
PaymentID - identyfikator płatności.
ConferenceReservationID - identyfikator rezerwacji, za którą wykonano płatność
PaymentDay - dzień płatności

```
CREATE TABLE Payment (PaymentID int IDENTITY(1,1) NOT NULL,
ConferenceReservationID int NOT NULL UNIQUE, PaymentDay date NOT NULL, PRIMARY KEY
(PaymentID));
```

```
ALTER TABLE Payment ADD CONSTRAINT FKPayment880455 FOREIGN KEY (ConferenceReservationID)
REFERENCES ConferenceReservation (ConferenceReservationID) on delete cascade;
```

3.9 PriceTreshold

Tabela przechowuje progi cenowe, zależne od wykonania daty płatności
PriceTresholdID - identyfikator progu cenowego
ConferenceID - identyfikator konferencji, którego dotyczą progi
beginDate - data od kiedy obowiązuje próg
Price - cena, która obowiązuje w danym okresie (za jeden dzień konferencji, za jednego uczestnika, studenci mają zniżkę 40%, ale wtedy wymagane jest podanie numeru legitymacji studenckiej i danych uczestnika danego dnia konferencji przed dokonaniem płatności, w innym wypadku zniżka nie zostanie naliczona)

```
CREATE TABLE PriceThreshold (PriceThresholdID int IDENTITY(1,1) NOT NULL,
ConferenceID int NOT NULL, beginDate date NOT NULL, Price int CHECK (Price >= 0 ) NOT NULL,
PRIMARY KEY (PriceThresholdID), UNIQUE (ConferenceID, beginDate));
```

```
ALTER TABLE PriceThreshold ADD CONSTRAINT FKPriceTresh540978 FOREIGN KEY (ConferenceID)
REFERENCES Conference (ConferenceID) on delete cascade ;
```

3.10 Workshops

Tabela przechowuje dane o warsztatach
WorkshopID - Identyfiaktor
DayID - Identyfikator dnia w którym odbywają się warsztaty
StartTime - Czas rozpoczęcia warsztatów
EndTime - Czas zakończenia warsztatów
NumberOfPlaces - Ilosć miejsc na warsztacie
Price - Cena za warsztat (jest stała, nie ma zniżek dla studentów, warsztat może być darmowy)

```
CREATE TABLE Workshops (WorkshopID int IDENTITY(1,1) NOT NULL, Name varchar(255) NOT NULL,
DayID int NOT NULL,
StartTime time NOT NULL, EndTime time NOT NULL, NumberOfPlaces int CHECK(NumberOfPlaces > 0)
NOT NULL, Price int CHECK(Price >= 0) NULL, PRIMARY KEY (WorkshopID),
CONSTRAINT CHK_TIME CHECK (StartTime < EndTime ));
```

```
ALTER TABLE Workshops ADD CONSTRAINT FKWorkshops622279 FOREIGN KEY (DayID) REFERENCES Days
(DayID) on delete cascade;
```

3.11 WorkshopsParticipants

Tabela przechowuje informacje o uczestnikach danego warsztatu, zgłoszonych przez klienta.
WorkshopsParticipantsID - identyfikator uczestnika warsztatu.
ParticipantID- identyfikator uczestnika.
WorkShopReservationID - identyfikator rezerwacji, na podstawie której zgłoszony jest uczestnik do udziału w warsztacie

```
CREATE TABLE WorkshopsParticipants (WorkshopsParticipantsID int IDENTITY(1,1) NOT NULL,
ParticipantForDayID int NOT NULL, WorkShopReservationID int NOT NULL, PRIMARY KEY
(WorkshopsParticipantsID), UNIQUE (ParticipantForDayID, WorkShopReservationID));
```

```
ALTER TABLE WorkshopsReservation ADD CONSTRAINT FKWorkshopsR574165 FOREIGN KEY
(DayReservationID) REFERENCES DayReservation (DayReservationID);
ALTER TABLE WorkshopsReservation ADD CONSTRAINT FKWorkshopsR114786 FOREIGN KEY (WorkshopID)
REFERENCES Workshops (WorkshopID) on delete cascade;
```

3.12 WorkshopsReservation

Przechowuje informacje o rezerwacji warsztatu na dany dzień
WorkshopsReservationID - Identyfikator rezerwacji
DayReservationID - Rezerwacji, którego dnia dotyczy
WorkshopID - Identyfikator rezerwowanego warsztatu
NumberOfPlaces - Ilość zarezerwowanych miejsc na warsztat

```
CREATE TABLE WorkshopsReservation (WorkshopsReservationID int IDENTITY(1,1) NOT NULL,
DayReservationID int NOT NULL, WorkshopID int NOT NULL, NumberOfPlaces int
CHECK(NumberOfPlaces > 0) NOT NULL, PRIMARY KEY (WorkshopsReservationID), UNIQUE
(WorkshopID, DayReservationID));
```

```
ALTER TABLE WorkshopsReservation ADD CONSTRAINT FKWorkshopsR574165 FOREIGN KEY
(DayReservationID) REFERENCES DayReservation (DayReservationID);
ALTER TABLE WorkshopsReservation ADD CONSTRAINT FKWorkshopsR114786 FOREIGN KEY (WorkshopID)
REFERENCES Workshops (WorkshopID) on delete cascade;
```

3.13 Usuwanie Tabel

```
ALTER TABLE WorkshopsParticipants DROP CONSTRAINT FKWorkshopsP170974;
ALTER TABLE WorkshopsParticipants DROP CONSTRAINT FKWorkshopsP739489;
ALTER TABLE Payment DROP CONSTRAINT FKPayment880455;
ALTER TABLE PriceTreshold DROP CONSTRAINT FKPriceTresh540978;
ALTER TABLE ParticipantsForDay DROP CONSTRAINT FKParticipan276571;
ALTER TABLE WorkshopsReservation DROP CONSTRAINT FKWorkshopsR574165;
ALTER TABLE WorkshopsReservation DROP CONSTRAINT FKWorkshopsR114786;
ALTER TABLE DayReservation DROP CONSTRAINT FKDayReserva734297;
ALTER TABLE DayReservation DROP CONSTRAINT FKDayReserva96605;
ALTER TABLE ParticipantsForDay DROP CONSTRAINT FKParticipan37239;
ALTER TABLE ConferenceReservation DROP CONSTRAINT FKConference392607;
ALTER TABLE ConferenceReservation DROP CONSTRAINT FKConference595044;
ALTER TABLE Participant DROP CONSTRAINT FKParticipan762161;
ALTER TABLE Days DROP CONSTRAINT FKDays575639;
ALTER TABLE Workshops DROP CONSTRAINT FKWorkshops622279;
DROP TABLE Company;
DROP TABLE Conference;
DROP TABLE ConferenceReservation;
DROP TABLE DayReservation;
DROP TABLE Days;
DROP TABLE Participant;
DROP TABLE ParticipantsForDay;
DROP TABLE Payment;
DROP TABLE PriceTreshold;
DROP TABLE Workshops;
DROP TABLE WorkshopsParticipants;
DROP TABLE WorkshopsReservation;
```

4. Widoki

4.1 UnpaidReservations

Widok przedstawia rezerwacje konferencji, które nie zostały jeszcze opłacone, wraz z danymi Klientów, którzy jej dokonali.

```
create view  UnpaidReservations as
   select c.ConferenceReservationID, c.ReservationDate, com.CompanyName, com.Address,
   com.Phone, com.[e-mail], DATEDIFF(day, c.ReservationDate, getdate()) as
DaysFromReservationsDate
   from ConferenceReservation as c inner join Company as com on c.CompanyID = com.CompanyID
   left outer join Payment as p on p.ConferenceReservationID = c.ConferenceReservationID
   where p.PaymentID is null
```

4.2 ConferenceDayParticipants

Widok przedstawia dane personalne uczestników zapisanych na dany dzień konferencji.

```
CREATE VIEW ConferenceDayParticipants AS
  SELECT
C3.Name,d.DayDate,p2.FirstName,p2.LastName,p2.StudentCard,c4.Address,C4.CompanyName,C4.Phon
e,C4.[e-mail] from Days d join Conference C3 on d.ConferenceID = C3.ConferenceID
  JOIN DayReservation dr on d.DayID = dr.DayID
  JOIN ParticipantsForDay pfr on pfr.DayReservationID = dr.DayReservationID
  JOIN Participant P2 on pfr.ParticipantID = P2.ParticipantID
  JOIN Company C4 on P2.CompanyID = C4.CompanyID
```

4.3 ConferenceWorkshopParticipants

Widok przedstawia dane personalne uczestników zapisanych na dany warsztat podczas konferencji.

```
CREATE VIEW ConferenceWorkshopParticipants AS
 SELECT C3.Name,w.Name as
WorkshopName,d.DayDate,w.StartTime,w.EndTime,w.NumberOfPlaces,w.Price,p2.FirstName,p2.LastN
ame,p2.StudentCard,c4.Address,C4.CompanyName,C4.Phone,C4.[e-mail] from Days d join
Conference C3 on d.ConferenceID = C3.ConferenceID
 JOIN Workshops W on d.DayID = W.DayID
 JOIN WorkshopsReservation wr on wr.WorkshopID = w.WorkshopID
 join WorkshopsParticipants wp on wp.WorkShopReservationID = wr.WorkshopsReservationID
 JOIN ParticipantsForDay pfr on pfr.ParticipantsForDayID = wp.ParticipantForDayID
 JOIN Participant P2 on pfr.ParticipantID = P2.ParticipantID
 JOIN Company C4 on P2.CompanyID = C4.CompanyID
```

4.4 MostActiveCustomers

Widok przedstawia dane klientów, którzy uczestniczyli w największej ilości konferencji (wraz z liczbą konferencji, w których uczestniczyli).

```
CREATE VIEW MostActiveCustomers AS
   SELECT C.Address,C.CompanyName,c.Phone,C.[e-mail],count(*) as conferences FROM Company C
   JOIN ConferenceReservation CR3 on C.CompanyID = CR3.CompanyID
   GROUP BY cr3.CompanyID,C.Address, C.CompanyName, c.Phone, C.[e-mail]
```

4.5 CustomersWithNotFilledReservation

Widok przedstawia klientów, którzy nie podali, która osoba zapisana jest na jaki dzień konferencji.

```
create view CustomersWithNotFilledReservation as
   select c.CompanyID, c.Address, c.CompanyName, c.Phone, c.[e-mail],
cr.ConferenceReservationID
   from Company as c inner join ConferenceReservation as cr on c.CompanyID = cr.CompanyID
   where cr.ConferenceReservationID in
   ( select dr.ConferenceReservationID from DayReservation as dr left outer join
ParticipantsForDay as p on p.DayReservationID = dr.DayReservationID
       group by dr.ConferenceReservationID, dr.DayReservationID, dr.NumberOfPlaces
       having dr.NumberOfPlaces > count(p.ParticipantsForDayID)
   )
   or cr.ConferenceReservationID in
   (
       select dr2.ConferenceReservationID from DayReservation as dr2 inner join
WorkshopsReservation as wr on dr2.DayReservationID = wr.DayReservationID
       left outer join WorkshopsParticipants as wp on wp.WorkShopReservationID =
wr.WorkshopsReservationID
       group by dr2.ConferenceReservationID, wr.WorkshopsReservationID, wr.NumberOfPlaces
       having wr.NumberOfPlaces > count(wp.WorkshopsParticipantsID)
       )
```

4.6 FreePlacesPerConferenceDay

Widok wypisuje ile jest jeszcze wolnych miejsc na dany dzień konferencji.

```
CREATE VIEW FreePlacesPerConferenceDay AS
 SELECT D.DayDate,Name,(dbo.DayFreePlaces(D.DayID)) AS freeplaces from Days d
   join Conference C4 on d.ConferenceID = C4.ConferenceID
```

4.7 FreePlacesPerConferenceWorkshop

Widok wypisuje ile jest jeszcze wolnych miejsc na dany warsztat.

```
CREATE VIEW FreePlacesPerConferenceDay AS
 SELECT D.DayDate,Name,(dbo.DayFreePlaces(D.DayID)) AS freeplaces from Days d
   join Conference C4 on d.ConferenceID = C4.ConferenceID
```

4.8 ConferencePopularity

Widok przedstawia konferencje wraz z ilością uczestników.

```
create view ConferencePopularity as
   select c.ConferenceID, c.BeginDate, c.EndDate, c.Name,
dbo.ConferenceAttendance(c.ConferenceID) as Attendance from Conference as c
```

4.9 ConferenceIncome

Widok przedstawia konferencje wraz z dochodem jakie wygenerowały.

```
create view ConferenceIncome as
   select c.ConferenceID, c.BeginDate, c.EndDate, c.Name,
sum(dbo.CalculateAmountToPay(cr.ConferenceReservationID)) as ConferenceIncome
   from Conference as c left outer join ConferenceReservation as cr on c.ConferenceID =
cr.ConferenceID
   group by c.ConferenceID, c.BeginDate, c.EndDate, c.Name
```

4.10 WorkshopsPopularity

Widok przedstawia warsztaty wraz z ilością uczestników.

```
create view WorkshopsPopularity as
   select w.WorkshopID, w.Name, w.DayID, w.StartTime, w.EndTime, w.NumberOfPlaces, w.Price,
coalesce(sum(wr.NumberOfPlaces), 0) as NumberOfParticipants
   from Workshops as w left outer join WorkshopsReservation as wr on w.WorkshopID =
wr.WorkshopID
   group by w.WorkshopID, w.WorkshopID, w.Name, w.DayID, w.StartTime, w.EndTime,
w.NumberOfPlaces, w.Price
go
```

4.11 MostActiveParticipants

Widok przedstawia uczestników, którzy uczestniczyli w największej liczbie konferencji

```
CREATE VIEW MostActiveParticipants AS
 SELECT
u.FirstName,u.LastName,u.StudentCard,c3.[e-mail],c3.Phone,c3.Address,c3.CompanyName,count(*
) as Conferences from Participant u
   JOIN Company C3 on u.CompanyID = C3.CompanyID
   JOIN ConferenceReservation CR on C3.CompanyID = CR.CompanyID
   where u.ParticipantID in (select pd.ParticipantID from ParticipantsForDay as pd inner
join DayReservation DR on pd.DayReservationID = DR.DayReservationID
       inner join ConferenceReservation C on DR.ConferenceReservationID =
C.ConferenceReservationID where C.ConferenceReservationID = CR.ConferenceReservationID)
   group by u.ParticipantID,u.FirstName, u.LastName, u.StudentCard, c3.[e-mail], c3.Phone,
c3.Address, c3.CompanyName, c3.CompanyID
```

4.12 PaidReservation

Widok przedstawia rezerwacje, wraz z danymi klientów, którzy opłacili swój udział

```
create view  PaidReservations as
   select c.ConferenceReservationID, c.ReservationDate, com.CompanyName, com.Address,
   com.Phone, com.[e-mail]
   from ConferenceReservation as c inner join Company as com on c.CompanyID = com.CompanyID
   left outer join Payment as p on p.ConferenceReservationID = c.ConferenceReservationID
   where p.PaymentID is not null
```

4.13 AmountToPayForConferenceReservation

Widok przedstawia rezerwacje konferencji, wraz z danymi firmy i kwotą do zapłaty.

```
CREATE VIEW AmountToPayForConferenceReservation AS
SELECT
Conference.Name,CompanyName,Address,Phone,[e-mail],dbo.CalculateAmountTo
Pay(cr.ConferenceReservationID) as toPay from Conference join
ConferenceReservation CR on Conference.ConferenceID = CR.ConferenceID
join Company C on CR.CompanyID = C.CompanyID
```

4.14 OneParticipantCompanies

Widok przedstawia jednoosobowe firmy to jest osoby prywatne,pokazuje te osoby.

```
CREATE VIEW OneParticipantCompanies AS
   SELECT
CompanyName,Address,Phone,[e-mail],FirstName,LastName,StudentCard from
Company c join Participant P on c.CompanyID = P.CompanyID
where c.CompanyID in(SELECT Company.CompanyID from Company join
Participant P on Company.CompanyID = P.CompanyID
group by p.CompanyID,Company.CompanyID
having count(*) = 1)
```

4.15 ManyParticipantsCompanies

Widok przedstawia firmy z co najmniej dwoma pracownikami, dla każdej firmy pokazuje liczbę pracowników.

```
CREATE VIEW ManyParticipantsCompanies AS
   SELECT CompanyName,Address,Phone,[e-mail],count(*) as employees from
Company c join Participant P on c.CompanyID = P.CompanyID
GROUP BY CompanyName, Address, Phone, [e-mail] HAVING count(*) > 1
```

4.16 ParticipantsWithCompanies
Widok przedstawia pracowników wraz ich firmami.

```
CREATE VIEW ParticipantsWithCompanies AS
  SELECT
FirstName,LastName,StudentCard,CompanyName,Address,Phone,[e-mail] from
Company c join Participant P on c.CompanyID = P.CompanyID
```

4.17 StudentsParticipants
Widok przedstawia uczestników którzy są studentam.

```
CREATE VIEW StudentsParticipants AS
  SELECT
FirstName,LastName,StudentCard,CompanyName,Address,Phone,[e-mail] from
Company c join Participant P on c.CompanyID = P.CompanyID
where StudentCard is not null
```

4.18 NotStudentsParticipants
Widok przedstawia uczestników którzy nie są studentam.

```
CREATE VIEW NotStudentsParticipants AS
  SELECT
FirstName,LastName,StudentCard,CompanyName,Address,Phone,[e-mail] from
Company c join Participant P on c.CompanyID = P.CompanyID
where StudentCard is  null
```

5 . Procedury

5.1 AddConference
Dodaje Konferencje.

```
create procedure AddConference  @name varchar(255), @beginDate date, @endDate date as
BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
       IF(@beginDate < GETDATE())
       BEGIN
           ;THROW 52000,
           'Konferencje nie moga byc tworzone w przeszlosci', 1;
       END
        IF(@beginDate > @endDate)
       BEGIN
           ;THROW 52000,
           'Data rozpoczęcia konferencji musi byc przed jej zakończeniem', 1;
```

```
        END
        INSERT INTO Conference(begindate, enddate, name) VALUES (@beginDate, @endDate, @name)
    END TRY
    BEGIN CATCH
        DECLARE @msg NVARCHAR(2048) =
        'Bład stworzenia konferencji:' +
        CHAR(13) + CHAR(10) + ERROR_MESSAGE();
        THROW 52000,@msg, 1;
    END CATCH
END
```

5.2 UpdateConferenceDayCapacity

Ustawia liczbę miejsc na dany dzień konferencji. (jeśli jest już zapisanych więcej uczestników to nie można zmienić na mniej - rzuca wyjatek)

```
create procedure UpdateConferenceDayCapacity @DayID int,@NewCapacity int as
BEGIN
  SET NOCOUNT ON;
  BEGIN TRY
      IF NOT EXISTS
      (
          SELECT * FROM Days
           WHERE DayID = @DayID
       )
       BEGIN
          ; THROW 52000 , 'Podany dzien konferencji nie istnieje' ,1
       END
      UPDATE Days set NumberOfPlaces=@NewCapacity where DayID=@DayID
  END TRY
  BEGIN CATCH
       DECLARE @errorMsg nvarchar (2048)
       = 'Cannot Update Day Capacity. Error message : '
       + ERROR_MESSAGE () ;
       ; THROW 52000 , @errorMsg ,1
  END CATCH
END
```

5.3 AddWorkshop

Dodaje warsztat do danego dnia konferencji.

```
CREATE PROCEDURE AddWorkshop @DayID int , @Name nvarchar (255) , @NumberOfPlaces int ,
@StartHour time (7) , @EndHour time (7) , @Price money AS
BEGIN
    SET NOCOUNT ON
   BEGIN TRY
        IF NOT EXISTS
        (
            SELECT * FROM Days
            WHERE DayID = @DayID
        )
        BEGIN
            ; THROW 52000 , 'Nie istnieje podany dzień konferencji.' ,1
        END
        INSERT INTO Workshops
        (
            DayID,
            Name,
            NumberOfPlaces ,
            StartTime,
            EndTime ,
            Price
        )
        VALUES
        (
        @DayID ,
        @Name ,
        @NumberOfPlaces ,
        @StartHour ,
        @EndHour ,
        @Price
        )
   END TRY
    BEGIN CATCH
        DECLARE @errorMsg nvarchar (2048)
        = 'Cannot add workshop . Error message : '
        + ERROR_MESSAGE () ;
        ; THROW 52000 , @errorMsg ,1
    END CATCH
END
```

5.4 RemoveConference

Usuwa konferencje i kaskadowo, wszystkie powiązane z nią dane.

```
create procedure RemoveConference   @id int as
BEGIN
   BEGIN TRY
       IF NOT EXISTS
       (
           SELECT * FROM Conference
           WHERE ConferenceID = @id
       )
       BEGIN
           ; THROW 52000 , 'Konferencja nie istnieje',1
       END
       delete from Conference where ConferenceID = @id
   END TRY
   BEGIN CATCH
       DECLARE @errorMsg nvarchar (2048)
       = 'Cannot delete price treshold . Error message : '
       + ERROR_MESSAGE () ;
       ; THROW 52000 , @errorMsg ,1;
   END CATCH
END
```

5.5 RemoveWorkshop

Usuwa warsztat i kaskadowo wszystkie powiązane z nim dane.

```
create procedure RemoveWorkshop @WorkshopID int as
BEGIN
   BEGIN TRY
       IF NOT EXISTS
       (
           SELECT * FROM Workshops
            WHERE WorkshopID = @WorkshopID
        )
        BEGIN
           ; THROW 52000 , 'Warsztat nie istnieje' ,1
        END
        DELETE Workshops
        WHERE WorkshopID = @WorkshopID
   END TRY
    BEGIN CATCH
        DECLARE @errorMsg nvarchar (2048)
        = 'Cannot delete Workshop. Error message : '
        + ERROR_MESSAGE () ;
        ; THROW 52000 , @errorMsg ,1
    END CATCH
END
```

5.6 AddPriceThreshold

Dodaje próg cenowy (Wymaga podania od kiedy dany próg obowiązuje)

```
CREATE PROCEDURE AddPriceThreshold @ConferenceID int , @beginDate date , @Price int AS
BEGIN
   SET NOCOUNT ON
  BEGIN TRY
       IF NOT EXISTS
       (
           SELECT * FROM Conference
           WHERE ConferenceID = @ConferenceID
       )
       BEGIN
           ; THROW 52000 , 'Nie istnieje podana konferencja.' ,1
       END
       INSERT INTO PriceThreshold
       (
           ConferenceID,
           beginDate,
           Price
       )
       VALUES
       (
       @ConferenceID ,
       @beginDate ,
       @Price
       )
  END TRY
   BEGIN CATCH
       DECLARE @errorMsg nvarchar (2048)
       = 'Cannot add PriceThreshold . Error message : '
       + ERROR_MESSAGE () ;
       ; THROW 52000 , @errorMsg ,1
   END CATCH
END
```

5.7 RemovePriceThreshold

Usuwa próg cenowy.

```
create procedure RemovePriceThreshold @PriceThresholdID int as
BEGIN
  BEGIN TRY
      IF NOT EXISTS
      (
```

```
          SELECT * FROM PriceThreshold
           WHERE PriceThresholdID = @PriceThresholdID
       )
       BEGIN
          ; THROW 52000 , 'Próg cenowy nie istnieje' ,1
       END
       DELETE PriceThreshold
       WHERE PriceThresholdID = @PriceThresholdID
  END TRY
   BEGIN CATCH
       DECLARE @errorMsg nvarchar (2048)
       = 'Cannot delete PriceThreshold. Error message : '
       + ERROR_MESSAGE () ;
       ; THROW 52000 , @errorMsg ,1
   END CATCH
END
```

5.8 UpdateWorkshopCapacity

Zmienia liczbę miejsc na warsztat. (Jeśli jest już zapisanych więcej uczestników, procedura nie wykona się i rzuca wyjątek)

```
create procedure UpdateWorkshopCapacity @WorkshopID int,@NewCapacity int as
BEGIN
  SET NOCOUNT ON;
  BEGIN TRY
      IF NOT EXISTS
      (
          SELECT * FROM Workshops
           WHERE WorkshopID = @WorkshopID
       )
       BEGIN
          ; THROW 52000 , 'Warsztat nie istnieje' ,1
       END
      UPDATE Workshops set NumberOfPlaces=@NewCapacity where WorkshopID=@WorkshopID
  END TRY
  BEGIN CATCH
       DECLARE @errorMsg nvarchar (2048)
       = 'Cannot Update Workshop Capacity. Error message : '
       + ERROR_MESSAGE () ;
       ; THROW 52000 , @errorMsg ,1
  END CATCH
END
```

5.9 AddCompany

Dodaje nowego klienta.

```
CREATE PROCEDURE AddCompany @Address varchar (255) , @CompanyName varchar (255) , @Phone
varchar(127), @email varchar (63) AS
BEGIN
   SET NOCOUNT ON
  BEGIN TRY
       INSERT INTO Company
       (
           Address,
           CompanyName,
           Phone,
           [e-mail]

       )
       VALUES
       (
       @Address ,
       @CompanyName ,
       @Phone,
       @email
       )
  END TRY
   BEGIN CATCH
       DECLARE @errorMsg nvarchar (2048)
       = 'Cannot add Comapny . Error message : '
       + ERROR_MESSAGE () ;
       ; THROW 52000 , @errorMsg ,1
   END CATCH
END
```

5.10 BookConference

Dodaje rezerwacje konferencji.

```
CREATE PROCEDURE BookConference @CompanyID int, @ConferenceID int AS
BEGIN
   SET NOCOUNT ON
   DECLARE @Date date = GETDATE()
   Declare @ConferenceDate date =
   (
       select BeginDate from Conference where ConferenceID = @ConferenceID
   )
    BEGIN TRY
        IF NOT EXISTS
        (
```

```
            SELECT * FROM Conference
            WHERE ConferenceID = @ConferenceID
        )
        BEGIN
            ; THROW 52000 , 'Konferencja nie istnieje.',1
        END
        IF NOT EXISTS
        (
            SELECT * FROM Company
            WHERE CompanyID = @CompanyID
        )
        BEGIN
            ; THROW 52000 , 'Firma nie istnieje.' ,1
        END
        IF @ConferenceDate < ( DATEADD (day , 14 , @Date ) )
        BEGIN
            ; THROW 52000 , 'Nie mozna zarezerwować konferencji na mniej niz 14 dni przed
jej rozpoczęciem' ,1
        END
        INSERT INTO ConferenceReservation
        (
            CompanyID,
            ConferenceID ,
            ReservationDate
        )
        VALUES
        (
             @CompanyID ,
             @ConferenceID ,
             @Date
        )
    END TRY
    BEGIN CATCH
        DECLARE @errorMsg nvarchar (2048)
        = 'Cannot book conference . Error message : '
        + ERROR_MESSAGE () ;
        ; THROW 52000 , @errorMsg ,1;
    END CATCH
END
```

5.11 BookDay

Dodaje rezerwacje dnia (jeśli nie ma miejsc rzuca wyjątek).

```
CREATE PROCEDURE BookDay @ConferenceReservationID int , @DayID int , @NumberOfPlaces int
AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
        IF NOT EXISTS
       (
            SELECT * FROM ConferenceReservation
            WHERE ConferenceReservationID = @ConferenceReservationID
        )
        BEGIN
           ; THROW 52000 , 'Rezerwacja konferencji nie istnieje.' ,1
        END
        IF NOT EXISTS
        (
        SELECT * FROM Days
        WHERE DayID = @DayID
        )
        BEGIN
           ; THROW 52000 , 'Dzień nie istnieje.' ,1
        END
        INSERT INTO DayReservation
        (
           DayID ,
           ConferenceReservationID ,
           NumberOfPlaces
        )
        VALUES
        (
            @DayID ,
            @ConferenceReservationID ,
            @NumberOfPlaces
        )
    END TRY
    BEGIN CATCH
        DECLARE @errorMsg nvarchar (2048)
        = 'Cannot book conference day. Error message : '
        + ERROR_MESSAGE () ;
        ; THROW 52000 , @errorMsg ,1;
    END CATCH
END
```

5.12 BookWorkshop

Dodaje rezerwacje warsztatu (jeśli nie ma miejsc rzuca wyjątek).

```
CREATE PROCEDURE BookWorkshop @DayReservationID int, @WorkshopID int, @NumberOfPlaces
int as
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
       IF NOT EXISTS
        (
            SELECT * FROM Workshops
            WHERE WorkshopID = @WorkshopID
        )
        BEGIN
           ; THROW 52000 , 'Podany warsztat nie istnieje.' ,1
        END
        IF NOT EXISTS
        (
            SELECT * FROM DayReservation
            WHERE DayReservationID = @DayReservationID
        )
        BEGIN
           ; THROW 52000 , 'Podana rezerwacja dnia nie istnieje.' ,1
        END
        INSERT INTO WorkshopsReservation
        (
           DayReservationID,
           WorkshopID ,
           NumberOfPlaces
        )
        VALUES
        (
            @DayReservationID ,
            @WorkshopID ,
            @NumberOfPlaces
        )
    END TRY
    BEGIN CATCH
        DECLARE @errorMsg nvarchar (2048)
        = 'Cannot book workshop . Error message : '
        + ERROR_MESSAGE () ;
        ; THROW 52000 , @errorMsg ,1;
    END CATCH
END
```

5.13 CancelConferenceReservation

Klient może anulować rezerwację, wtedy usuwają się kaskadowo wszystkie związane z nią dane. (musi wywołać CancelDayReservation dla każdego dnia powiązanego)

```
CREATE PROCEDURE CancelUnpaidOnTimeReservations as
BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
       DECLARE @CurrentID INT = 0
       -- Iterate over all customers
       WHILE (1 = 1)
       BEGIN
             -- Get next customerId
             SELECT TOP 1 @CurrentID = c.ConferenceReservationID
             FROM ConferenceReservation as c
             WHERE c.ConferenceReservationID > @CurrentID
             ORDER BY c.ConferenceReservationID
             -- Exit loop if no more customers
             IF @@ROWCOUNT = 0 BREAK;
             -- call your sproc
             if @CurrentID in
                (
                SELECT cr.ConferenceReservationID
                FROM ConferenceReservation as cr
                left outer join Payment as p on p.ConferenceReservationID =
cr.ConferenceReservationID
                   where p.PaymentID is null and DATEDIFF(day,  cr.ReservationDate,
getdate()) > 7
                )
             begin
                 EXEC CancelConferenceReservation @ID = @CurrentID
             end
       END
    END TRY
    BEGIN CATCH
        DECLARE @errorMsg nvarchar (2048)
        = 'Cannot cancel unpaid time reservations . Error message : '
        + ERROR_MESSAGE () ;
        ; THROW 52000 , @errorMsg ,1;
    END CATCH
END
```

5.14 CancelDayReservation

Klient może anulować rezerwację na dany dzień, nastąpi kaskadowe usunięcie powiązanych danych. (musi wywołać CancelWorkshopReservation dla każdego powiązanego warsztatu)

```
CREATE PROCEDURE CancelDayReservation @DayReservationID int as
BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
       IF NOT EXISTS
       (
       SELECT * FROM DayReservation
        WHERE DayReservationID = @DayReservationID
        )
        BEGIN
           ; THROW 52000 , 'Rezerwacja dnia nie istnieje.' ,1
        END
       DELETE WorkshopsParticipants where WorkShopReservationID in (select
wr.WorkshopsReservationID from WorkshopsReservation as wr where wr.DayReservationID =
@DayReservationID)
       DELETE WorkshopsReservation where DayReservationID = @DayReservationID
       DELETE DayReservation where DayReservationID = @DayReservationID
    END TRY
    BEGIN CATCH
        DECLARE @errorMsg nvarchar (2048)
        = 'Cannot cancel Day Reservation. Error message : '
        + ERROR_MESSAGE () ;
        ; THROW 52000 , @errorMsg ,1;
    END CATCH
END
```

5.15 CancelWorkshopReservation

Anulowanie rezerwacji warsztatu, skutkuje kaskadowym usunięciem powiązanych danych. (musi wywołać RemoveParticpantFromWorkshop dla każego powiązanego uczestnika)

```
CREATE PROCEDURE CancelWorkshopReservation @WorkshopReservationID int as
BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
       IF NOT EXISTS
       (
       SELECT * FROM WorkshopsReservation
        WHERE WorkshopsReservationID = @WorkshopReservationID
        )
```

```
        BEGIN
           ; THROW 52000 , 'Rezerwacja warsztatu ie istnieje.' ,1
        END
       DELETE WorkshopsParticipants where WorkShopReservationID = @WorkshopReservationID
       DELETE WorkshopsReservation where WorkshopsReservationID = @WorkshopReservationID
    END TRY
    BEGIN CATCH
        DECLARE @errorMsg nvarchar (2048)
        = 'Cannot cancel Workshop Reservation. Error message : '
        + ERROR_MESSAGE () ;
        ; THROW 52000 , @errorMsg ,1;
    END CATCH
END
```

5.16 CancelUnpaidOnTimeReservation

Wyszukuje nieopłaconych rezerwacji i je usuwa, wraz z powiązanymi danymi.

```
CREATE PROCEDURE CancelUnpaidOnTimeReservations as
BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
       select dbo.CancelConferenceReservation(c.ConferenceReservationID) from
ConferenceReservation as c where c.ConferenceReservationID in
        (
        SELECT cr.ConferenceReservationID
        FROM ConferenceReservation as cr
        left outer join Payment as p on p.ConferenceReservationID =
cr.ConferenceReservationID
           where p.PaymentID is null and DATEDIFF(day, getdate(), cr.ReservationDate) > 7
        )
    END TRY
    BEGIN CATCH
        DECLARE @errorMsg nvarchar (2048)
        = 'Cannot cancel unpaid time reservations . Error message : '
        + ERROR_MESSAGE () ;
        ; THROW 52000 , @errorMsg ,1;
    END CATCH
END
```

5.17 AddPayment

Dodaje płatność za rezerwację.

```
create procedure AddPayment  @ConferenceReservationID int as
BEGIN
  SET NOCOUNT ON;
  BEGIN TRY
      DECLARE @Date date = getDate();
      IF NOT EXISTS
      (
          SELECT * FROM ConferenceReservation
           WHERE ConferenceReservationID = @ConferenceReservationID
       )
       BEGIN
          ; THROW 52000 , 'Rezerwacja nie istnieje' ,1
       END
      INSERT INTO Payment(ConferenceReservationID,PaymentDay) VALUES
(@ConferenceReservationID, @Date)
  END TRY
  BEGIN CATCH
      DECLARE @msg NVARCHAR(2048) =
      'Bład stworzenia płatności:' +
      CHAR(13) + CHAR(10) + ERROR_MESSAGE();
      THROW 52000,@msg, 1;
  END CATCH
END
```

5.18 AddParticipantToDay

Dodaje uczestnika do danego dnia konferencji. (jeśli firma wykorzystała już wszystkie zadeklarowane miejsca to nie może dodać kolejnego uczestnika, rzucany jest wyjątek)

```
CREATE PROCEDURE AddParticipantToDay @ParticipantID int, @DayReservationID int as
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
       IF NOT EXISTS
        (
        SELECT * FROM DayReservation
        WHERE DayReservationID = @DayReservationID
        )
        BEGIN
           ; THROW 52000 , 'Nie istnieje podana rezerwacja dnia. ' ,1
        END
       IF NOT EXISTS
        (
        SELECT * FROM Participant
        WHERE ParticipantID = @ParticipantID
        )
```

```
        BEGIN
           ; THROW 52000 , 'Nie istnieje podany uczestnik. ' ,1
        END
        INSERT INTO ParticipantsForDay
        (
            DayReservationID,
            ParticipantID
        )
        VALUES
        (
           @DayReservationID,
           @ParticipantID
        )
    END TRY
    BEGIN CATCH
        DECLARE @errorMsg nvarchar (2048)
        = 'Cannot add participant to day . Error message : '
        + ERROR_MESSAGE () ;
        ; THROW 52000 , @errorMsg ,1;
    END CATCH
END
```

5.19 RemoveParticipantFromDay

Usuwa uczestnika z rezerwacji na dany dzień. Jeśli był on zapisany na warsztat w tym dniu również nastąpi usunięcie jego uczestnictwa w tym warsztacie.

```
CREATE PROCEDURE RemoveParticipantFromDay @ParticipantForDayID int as
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
       IF NOT EXISTS
       (
           SELECT * FROM ParticipantsForDay AS pd
            WHERE  pd.ParticipantsForDayID = @ParticipantForDayID
        )
        BEGIN
           ; THROW 50001 , 'Uczestnik ie był zapisany na ten dzień.' ,1
        END
        DELETE FROM ParticipantsForDay
        WHERE ParticipantsForDayID = @ParticipantForDayID
    END TRY
    BEGIN CATCH
        DECLARE @errorMsg nvarchar (2048)
        = 'Cannot remove Participant from day . Error message : '
        + ERROR_MESSAGE () ;
        ; THROW 52000 , @errorMsg ,1;
    END CATCH
```

```
END
```

5.20 AddParticipantToWorkshop

Dodaje uczestnika do rezerwacji warsztatu. (jeśli firma wykorzystała już wszystkie zadeklarowane miejsca to nie może dodać kolejnego uczestnika, rzucany jest wyjątek) (jeśli uczestnik jest zapisany na inny warsztat w tym samym czasie, rzucany jest wyjątek) (jeśli uczestnik nie jest zapisany na dzień w którym odbywa się warsztat rzuca wyjątek)

```
create procedure AddParticipantToWorkshop @ParticipantsForDayID
int,@WorkShopReservationId int as
BEGIN
 SET NOCOUNT ON;
 BEGIN TRY
     IF NOT EXISTS
     (
         SELECT * FROM ParticipantsForDay
          WHERE ParticipantsForDayID = @ParticipantsForDayID
      )
      BEGIN
         ; THROW 52000 , 'Uzytkownik nie jest zapisany na żaden dzien rezerwacji' ,1
      END
     IF NOT EXISTS
     (
         SELECT * FROM WorkshopsReservation
          WHERE WorkshopsReservationID = @WorkShopReservationId
      )
      BEGIN
         ; THROW 52000 , 'Podana rezewacja warsztatu nie istnieje' ,1
      END
     INSERT INTO WorkshopsParticipants(ParticipantForDayID,WorkShopReservationID) VALUES
(@ParticipantsForDayID, @WorkShopReservationID)
 END TRY
 BEGIN CATCH
   DECLARE @msg NVARCHAR(2048) =
   'Bład dodania uczestnika do warsztatu:' +
   ERROR_MESSAGE();
   THROW 52000,@msg, 1;
 END CATCH
END
go
```

5.21 RemoveParticipantFromWorkshop

Usuwa uczestnika z rezerwacji danego warsztatu.

```
CREATE PROCEDURE RemoveParticipantFromWorkshop @WorkshopParticipantID int as
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
       IF NOT EXISTS
       (
       SELECT * FROM WorkshopsParticipants AS p
        WHERE p.WorkshopsParticipantsID = @WorkshopParticipantID
        )
        BEGIN
            ; THROW 50001 , 'Podany uczestnik nie istnieje',1
        END
        DELETE FROM WorkshopsParticipants
        WHERE WorkshopsParticipantsID = @WorkshopParticipantID
    END TRY
    BEGIN CATCH
        DECLARE @errorMsg nvarchar (2048)
        = 'Cannot remove Workshop participant . Error message : '
        + ERROR_MESSAGE () ;
        ; THROW 52000 , @errorMsg ,1;
    END CATCH
END
```

5.22 AddParticipant

Tworzy profil uczestnika.

```
create procedure AddParticipant @CompanyID int,@FirstName varchar(63),@LastName
varchar(63),@StudentCard int as
BEGIN
  SET NOCOUNT ON;
  BEGIN TRY
      IF NOT EXISTS
      (
          SELECT * FROM Company
           WHERE CompanyID = @CompanyID
       )
       BEGIN
          ; THROW 52000 , 'Firma nie istnieje' ,1
       END
      INSERT INTO Participant(CompanyID,FirstName,LastName,StudentCard) VALUES
(@CompanyID, @FirstName,@LastName,@StudentCard)
  END TRY
```

```
  BEGIN CATCH
      DECLARE @msg NVARCHAR(2048) =
      'Bład stworzenia uczestnika:' +
      CHAR(13) + CHAR(10) + ERROR_MESSAGE();
      THROW 52000,@msg, 1;
  END CATCH
END
```

5.23 AddDay

Dodaje dzień konferencji. (sprawdza czy jest w czasie trwania konferencji)

```
create procedure AddDay @ConferenceId int, @NumberOfPlaces int, @DayDate date as
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
       IF NOT EXISTS
        (
            SELECT * FROM Conference
            WHERE ConferenceID = @ConferenceID
        )
        BEGIN
            ; THROW 52000 , 'Konferencja nie istnieje' ,1
        END
        INSERT INTO Days
        (
        ConferenceID, NumberOfPlaces, DayDate
        )
        VALUES
        (
        @ConferenceId ,
        @NumberOfPlaces,
        @DayDate
        )
    END TRY
    BEGIN CATCH
        DECLARE @errorMsg nvarchar (2048)
        = 'Cannot add conference day. Error message : '
        + ERROR_MESSAGE () ;
        ; THROW 52000 , @errorMsg ,1
   END CATCH
END
```

5.24 RemoveDay

Usuwa dzień konferencji i kaskadowo wszystkie powiązane z nim dane.

```
create procedure RemoveDay @DayID int as
BEGIN
   BEGIN TRY
       IF NOT EXISTS
       (
           SELECT * FROM Days
            WHERE DayID = @DayID
        )
        BEGIN
           ; THROW 52000 , 'Dzień nie istnieje' ,1
        END
        DELETE Days
        WHERE DayID = @DayID
   END TRY
    BEGIN CATCH
        DECLARE @errorMsg nvarchar (2048)
        = 'Cannot delete day. Error message : '
        + ERROR_MESSAGE () ;
        ; THROW 52000 , @errorMsg ,1
    END CATCH
END
```

6. Funkcje

6.1 CalculateAmountToPay

Dla danej rezerwacji konferencji, oblicza kwotę jaką trzeba zapłacić, sumuje opłatę za konferencję dla każdego uczestnika (sprawdza ile jest studentów) oraz opłatę za płatne warsztaty.

```
create function CalculateAmountToPay(@ConferenceReservationID int) RETURNS FLOAT as
begin
   DECLARE @Discount int = 0.4
   DECLARE @PricePerPersonPerDay int =
   (
       select top 1 pd.Price from PriceThreshold as pd
       inner join ConferenceReservation as cr on cr.ConferenceID = pd.ConferenceID
       where cr.ConferenceReservationID = @ConferenceReservationID and
             pd.beginDate < GETDATE()
       ORDER BY pd.beginDate DESC
   )
   return (
       (select coalesce(sum(priceForDay), 0) from
```

```
       (
       select
sum(dbo.CountStudentsNumber(dr.DayReservationID)*@PricePerPersonPerDay*@Discount +
(dr.NumberOfPlaces -
dbo.CountStudentsNumber(dr.DayReservationID))*@PricePerPersonPerDay) as priceForDay
       from DayReservation as dr where dr.ConferenceReservationID =
@ConferenceReservationID
       group by dr.DayReservationID, dr.NumberOfPlaces) as dayPrize )
       +
       (select coalesce(sum(pricePerWorkshop), 0) from (
           select sum(w.Price * wr.NumberOfPlaces) as pricePerWorkshop from Workshops as
w
           inner join WorkshopsReservation as wr on wr.WorkshopID = w.WorkshopID
           inner join DayReservation as dr on wr.DayReservationID = dr.DayReservationID
           where dr.ConferenceReservationID = @ConferenceReservationID
           group by w.WorkshopID, w.Price, wr.NumberOfPlaces
       ) as workshopPrice)
       )
end
```

6.2 DayFreePlaces

Zwraca ilość wolnych miejsc dla podanego dnia

```
CREATE FUNCTION DayFreePlaces ( @ID int) RETURNS integer as
begin

   declare @dayPlaces integer =
   (SELECT Days.NumberOfPlaces from Days
       where DayID = @ID)
   declare @reservPlaces integer =
   (SELECT sum(NumberOfPlaces) from DayReservation
       WHERE DayID = @ID group by DayID)
   IF @reservPlaces IS NULL
   begin
       return @dayPlaces
   end

   return @dayPlaces - @reservPlaces;
end
go
```

6.3 WorkshopFreePlaces

Zwraca liczbę miejsc dla danego warsztatu

```
CREATE FUNCTION WorkshopFreePlaces ( @ID int) RETURNS integer as
begin

   declare @dayPlaces integer =
   (SELECT Workshops.NumberOfPlaces from Workshops
       where WorkshopID = @ID)
   declare @reservPlaces integer =
   (SELECT sum(NumberOfPlaces) from WorkshopsReservation
       WHERE WorkshopID = @ID group by WorkshopID)

   IF (@reservPlaces is NULL)
   begin
       return @dayPlaces;
   end

   return @dayPlaces - @reservPlaces;
end
go
```

6.4 ConferenceDays

Zwraca identyfikatory dni danej konferencji

```
CREATE FUNCTION ConferenceDays ( @ID int) RETURNS TABLE
AS
   RETURN
   (
       SELECT d.DayID, d.DayDate, d.NumberOfPlaces
       FROM Days AS d
       where d.ConferenceID = @ID
    ) ;
```

6.5 WorkshopCollision

Zwraca wartość 1, jeśli dwa podane warsztaty odbywają się w tym samym czasie.

```
create function WorkshopCollision (@ID1 int, @ID2 int) RETURNS Bit as
begin
   DECLARE @Start_1 time = (select StartTime from Workshops where WorkshopID = @ID1);
   DECLARE @End_1 time = (select EndTime from Workshops where WorkshopID  = @ID1) ;
   DECLARE @Start_2 time = (select StartTime from Workshops where WorkshopID = @ID2) ;
   DECLARE @End_2 time =  (select EndTime from Workshops where WorkshopID  = @ID2) ;
   DECLARE @Day1 int = (select DayID from Workshops where WorkshopID = @ID1);
```

```
  DECLARE  @Day2 int = (select DayID from Workshops where WorkshopID = @ID2);
  IF @Day1 != @Day2
      RETURN 0
  IF @Start_1 < @Start_2 AND @Start_2 < @End_1
      RETURN 1
  IF @Start_2 < @Start_1 AND @Start_1 < @End_2
      RETURN 1
  IF @Start_1 >= @Start_2 AND @End_2 >= @End_1
      RETURN 1
  IF @Start_2 >= @Start_1 AND @End_1 >= @End_2
      RETURN 1
  RETURN 0
END
```

6.6 ConferenceFreePlaces

Zwraca ilość wolnych miejsc na wszystkie dni konferencji

```
CREATE FUNCTION ConferenceFreePlaces ( @ID int) RETURNS integer as
begin

   declare @ConferencePlaces integer =
   (select sum(dbo.DayFreePlaces(DayID))  from Days where ConferenceID = @ID)

   return @ConferencePlaces;
end
```

6.7 ConferenceAttendance

Zwraca ile jest uczestników konferencji przez wszystkie dni.

```
begin

  declare @ConferencePlaces integer =
  (select sum(Days.NumberOfPlaces)  from Days where ConferenceID = @ID)
   - (dbo.ConferenceFreePlaces(@ID))
  return @ConferencePlaces;
end
go
```

6.8 DayParticipantsList

Zwraca listę uczestników danego dnia konferencji

```
CREATE FUNCTION DayParticipantsList ( @ID int) RETURNS TABLE AS
  RETURN ( SELECT * from Participant where ParticipantID IN
      (SELECT ParticipantID FROM ParticipantsForDay where DayReservationID in
          (SELECT DayReservationID from DayReservation where
```

```
            DayID = @ID)));
```

6.9 WorkshopParticipantsList

Zwraca listę uczestników danego warsztatu.

```
CREATE FUNCTION WorkshopParticipantsList ( @ID int) RETURNS TABLE AS
  RETURN ( SELECT * from Participant where ParticipantID IN
      (SELECT ParticipantID FROM ParticipantsForDay where ParticipantsForDayID in
          (SELECT ParticipantForDayID from WorkshopsParticipants where
              WorkShopReservationID in
           (SELECT WorkshopsReservationID from WorkshopsReservation where WorkshopID =
@ID))))
go
```

6.10 ParticipantWorkshopsList

Zwraca listę warsztatów, w których bierze udział uczestnik.

```
CREATE FUNCTION ParticipantWorkshopsList ( @ID int) RETURNS TABLE AS
  RETURN ( SELECT * from Workshops where WorkshopID IN
      (SELECT WorkshopID FROM WorkshopsReservation where WorkshopsReservationID in
          (SELECT WorkShopReservationID from WorkshopsParticipants where
              ParticipantForDayID in
           (SELECT ParticipantsForDayID from ParticipantsForDay where ParticipantID =
@ID))));
```

6.11 CountStudentsNumber

Zwraca liczbę studentów zapisanych na dany dzień konferencji, dla danej rezerwacji.

```
create function CountStudentsNumber(@DayReservationID int) RETURNS INT as
begin
   return (select count(*) from ParticipantsForDay as pd
   inner join Participant as p on pd.ParticipantID = p.ParticipantID
   where pd.DayReservationID = @DayReservationID and p.StudentCard is not null)
end
```

6.12 DayWorkshops

Zwraca listę identyfikatorów warsztatów w danym dniu.

```
CREATE FUNCTION DayWorkshops (@ID int) RETURNS TABLE as
   RETURN
   (
       SELECT w.WorkshopID, w.Name, w.StartTime, w.EndTime, w.NumberOfPlaces, w.Price
       from Workshops as w
```

```
        where w.DayID = @ID
    )
```

6.13 ParticipantDayList

Zwraca listę dni, w których uczestniczy dany uczestnik danej konferencji.

```
create function ParticipantDayList(@ParticipantID int, @ConferenceID int) RETURNS TABLE as
    RETURN
    (
        select d.DayID, d.DayDate from Days as d inner join DayReservation as dr
        on dr.DayID = d.DayID inner join ParticipantsForDay as p on p.DayReservationID =
dr.DayReservationID
        where p.ParticipantID = @ParticipantID and d.ConferenceID = @ConferenceID
    )
```

7. Triggery

7.1 CorrectDayDate

Blokuje dodanie dnia konferencji, jeżeli jest on poza datą trwania konferencji.

```
CREATE TRIGGER CorrectDayDate ON Days AFTER INSERT AS
BEGIN
   SET NOCOUNT ON;
   IF EXISTS
   (
       SELECT * FROM inserted AS i
       inner join Conference AS c ON c.ConferenceID = i.ConferenceID
       WHERE i.DayDate < c.BeginDate or i.DayDate > c.EndDate
    )
    BEGIN
        ; THROW 50001 , 'Data dnia jest poza okresem trwania konferencji. ' ,1
    END
END
```

7.2 CorrectDayBooking

Sprawdza czy rezerwowany dzień, należy do konferencji, którą klient zarezerwował wcześniej.

```
create TRIGGER CorrectDayBooking on DayReservation after insert  as
begin
   set nocount on;
   IF EXISTS
```

```
   (
       SELECT *
        FROM inserted AS i
        inner join Days AS d
        ON d.DayID = i.DayID
        inner join Conference AS c1 ON c1.ConferenceID = d.ConferenceID
        inner join ConferenceReservation AS r
        ON i.ConferenceReservationID = r.ConferenceReservationID
        inner join Conference AS c2 ON c2.ConferenceID = r.ConferenceID
        WHERE c1.ConferenceID != c2.ConferenceID
    )
    BEGIN
        ; THROW 50001 , 'Rezerwowany dzień nie pochodzi z zarezerwowanej konferencji' ,1
    END
end
```

7.3 CorrectWorkshopBooking

Sprawdza czy aktualnie rezerwowany warsztat odbywa się w dniu, który zarezerwował klient.

```
create trigger CorrectWorkshopBooking on WorkshopsReservation after insert as
BEGIN
   SET NOCOUNT ON;
   IF EXISTS
   (
       SELECT *
        FROM inserted AS i
        inner join Workshops as w on i.WorkshopID = w.WorkshopID
        inner join Days AS d1
        ON d1.DayID = w.DayID
        inner join DayReservation as r on r.DayReservationID = i.DayReservationID
        inner join Days as d2 on d2.DayID = r.DayID
        WHERE d1.DayID != d2.DayID
    )
    BEGIN
        ; THROW 50001 , 'Warsztat odbywa sie w dniu, którego nie dotyczy podana rezerwacja
dnia' ,1
    END
END
```

7.4 NoPlaceForDay

Sprawdza, czy w zarezerwowanym dniu jest miejsce.

```
create TRIGGER NoPlaceForDay on DayReservation  after insert as
begin
```

```
  set nocount on;
  IF EXISTS
  (
      SELECT * from inserted as i
      where dbo.DayFreePlaces(i.DayID) < 0
   )
   BEGIN
       ; THROW 50001 , 'Nie ma wystarczającej ilosci wolnych miejsc w tym dniu' ,1
   END
end
```

7.5 NoPlaceForWorkshop

Sprawdza, czy w zarezerwowanym warsztacie jest miejsce.

```
create TRIGGER NoPlaceForWorkshop on WorkshopsReservation after insert  as
begin
  set nocount on;
  IF EXISTS
  (
      SELECT * from inserted as i
      where dbo.WorkshopFreePlaces (i.WorkshopID) < 0
   )
   BEGIN
       ; THROW 50001 , 'Nie ma wystarczającej ilosci wolnych miejsc na warsztat w tym dniu'
,1
   END
end
go
```

7.6 LessPlaceInWorkshopThanDay

Blokuje rezerwacje warsztatu, jeśli klient zadeklarował więcej miejsc niż w danym dniu.

```
CREATE TRIGGER LessPlaceInWorkshopThanDay on WorkshopsReservation AFTER INSERT , UPDATE AS
BEGIN
   SET NOCOUNT ON;
   IF EXISTS
   (
       SELECT * FROM inserted AS i
       inner join DayReservation as r on i.DayReservationID = r.DayReservationID
        WHERE i.NumberOfPlaces > r.NumberOfPlaces
    )
    BEGIN
        ; THROW 50001 , 'Nie możesz zarezerwować więcej miejsc na warsztat niż na dzień!' ,1
    END
END
```

7.7 NoPlaceForParticipantInDay

Blokuje dodanie uczestnika, jeżeli firma wykorzystała wszystkie zadeklarowane na dany dzień miejsca.

```
create trigger NoPlaceForParticipantInDay on ParticipantsForDay after insert as
begin
   SET NOCOUNT ON;
   IF EXISTS
   (
       SELECT * FROM inserted AS i
        WHERE ((select NumberOfPlaces from DayReservation where
DayReservation.DayReservationID = i.DayReservationID)
           < (select count(*) from ParticipantsForDay as p where p.DayReservationID =
i.DayReservationID))
   )
    BEGIN
       ; THROW 50001 , 'Nie ma miejsca dla kolejnego uczestnika na ten dzień konferencji.'
,1
    END
end
```

7.8 NoPlaceForParticipantInWorkshop

Blokuje dodanie uczestnika, jeżeli firma wykorzystała wszystkie zadeklarowane na dany warsztat miejsca.

```
create trigger NoPlaceForParticipantInWorkshop on WorkshopsParticipants after insert as
begin
  SET NOCOUNT ON;
  IF EXISTS
  (
      SELECT * FROM inserted AS i
       WHERE ((select NumberOfPlaces from WorkshopsReservation where
WorkshopsReservation.WorkshopsReservationID = i.WorkShopReservationID)
          < (select count(*) from WorkshopsParticipants as p where p.WorkShopReservationID
= i.WorkShopReservationID))
  )
   BEGIN
      ; THROW 50001 , 'Nie ma miejsca dla kolejnego uczestnika na ten warsztat.' ,1
   END
end
go
```

7.9 CorrectDayCapacity

Sprawdza, czy zmniejszona liczba uczestników na dany dzień jest większa od liczby już zapisanych uczestników.

```
CREATE TRIGGER CorrectDayCapacity on dbo.Days AFTER INSERT , UPDATE AS
BEGIN
  SET NOCOUNT ON;
  IF EXISTS
  (
     SELECT *
       FROM inserted AS i
      WHERE dbo.DayFreePlaces(i.DayID) < 0
   )
   BEGIN
       ; THROW 50001 , 'Na ten dzień konferencji zapisało się już więcej użydkowników niż
podana nowa liczba miejsc' ,1
   END
END
```

7.10 CorrectWorkshopCapacity

Sprawdza, czy zmniejszona liczba uczestników warsztatu jest większa od liczby już zapisanych uczestników.

```
CREATE TRIGGER CorrectWorkshopCapacity on dbo.Workshops AFTER INSERT , UPDATE AS
BEGIN
  SET NOCOUNT ON;
  IF EXISTS
  (
     SELECT *
       FROM inserted AS i
      WHERE dbo.WorkshopFreePlaces(i.WorkshopID) < 0
   )
   BEGIN
       ; THROW 50001 , 'Na warsztat zapisało się już więcej użydkowników niż podana nowa
liczba miejsc' ,1
   END
END
```

7.11 CheckWorkshopCollision

Blokuje dodanie uczestnka do warsztatu, jeżeli uczestniczy w warsztacie w tym samym czasie.

```
CREATE trigger CheckWorkshopCollision on WorkshopsParticipants after insert as
begin
   SET NOCOUNT ON;

       IF EXISTS(
           select * from inserted as i inner join WorkshopsReservation as wr
           on i.WorkShopReservationID = wr.WorkshopsReservationID
           inner join WorkshopsParticipants as wp on i.ParticipantForDayID =
wp.ParticipantForDayID
           inner join WorkshopsReservation as wr2 on wp.WorkShopReservationID =
wr2.WorkshopsReservationID
           where
           dbo.WorkshopCollision(wr2.WorkshopID, wr.WorkshopID) = 1
           and wr2.WorkshopID != wr.WorkshopID
       )
       BEGIN
        ; THROW 50001, 'Uzytkownik jest juz zapisny na inny warsztat w tym samym czasie', 1
       end
end
```

7.12 CanStudentBeAdded

Blokuje dodanie studenta jako uczestnika dnia konferencji, jeżeli dokonano już płatności za konferencje. Jeżeli chce się by student był uczestnikiem konferencji i skorzystać ze zniżek wymagane jest dodanie go jako uczestnika przy rezerwacji dnia, przed dokonaniem płatności.

```
create trigger CanStudentBeAdded on ParticipantsForDay after insert as
begin
   SET NOCOUNT ON;
   IF EXISTS
       (
select * from inserted as i inner join Participant as p on i.ParticipantID = p.ParticipantID
           inner join DayReservation as dr on dr.DayReservationID = i.DayReservationID
           inner join Payment as pay on pay.ConferenceReservationID =
dr.ConferenceReservationID
           where p.StudentCard is not null and pay.PaymentDay is not null
       )
   begin
        ; THROW 50001 , 'Rezerwacja została już opłacona, żeby dodać studenta i otrzymać za
niego zniżkę trzeba dokonać tego przy dokonaniu rezerwacji i przed dokonaniem płatności.
                        Jeżeli nie zrobi się tego w odpowiednim terminie, student może
uczestniczyć w konferencji ale jako zwykła osoba. ' ,1
   end
```

```
end
```

7.13 CorrectPriceThresholdDate

Blokuje dodanie progu cenowego, jeżeli jego data jest po rozpoczęciu konferencji.

```
CREATE TRIGGER CorrectPriceThresholdDate ON PriceThreshold AFTER INSERT AS
BEGIN
   SET NOCOUNT ON;
   IF EXISTS
   (
       SELECT * FROM inserted AS i
       inner join Conference AS c ON c.ConferenceID = i.ConferenceID
       WHERE i.beginDate > c.BeginDate
    )
    BEGIN
        ; THROW 50001 , 'Próg Cenowy rozpoczyna się po rozpoczęciu konferencji. ' ,1
    END
END
```

7.14 CorrectPartcipantCompany

Sprawdza, czy dodany do danego dnia uczestnik, jest pracownik firmy która wykonała rezerwacje.

```
create trigger CorrectPartcipantCompany on ParticipantsForDay after insert as
begin
   SET NOCOUNT ON;
   IF EXISTS
       (
           select * from inserted as i inner join dbo.Participant as p on i.ParticipantID =
p.ParticipantID
           inner join Company as c1 on p.CompanyID = c1.CompanyID inner join DayReservation
as dr on dr.DayReservationID = i.DayReservationID
           inner join ConferenceReservation as cr on dr.ConferenceReservationID =
cr.ConferenceReservationID
           inner join Company as c2 on cr.CompanyID = c2.CompanyID where c1.CompanyID !=
c2.CompanyID
       )
   begin
        ; THROW 50001 , 'Uczestnik nie jest pracownikiem firmy, która wykonała rezerwacje.'
,1
   end
end
```

7.15 CorrectWorkshopParticipantDay

Sprawdza, czy podana rezerwacja warsztatu pochodzi z tej samej rezerwacji dnia co podany uczestnik.

```
create trigger CorrectWorkshopParticipantDay on WorkshopsParticipants after insert as
begin
  SET NOCOUNT ON;
  IF  EXISTS(
      SELECT * from inserted as i
      inner join dbo.WorkshopsReservation as wr on i.WorkShopReservationID =
wr.WorkshopsReservationID
      inner join ParticipantsForDay as pd on i.ParticipantForDayID =
pd.ParticipantsForDayID
      where pd.DayReservationID != wr.DayReservationID
      )
   BEGIN
      ; THROW 50001 , 'Uzydkownik nie jest zapisany na dzien, którego dotyczy
rezerwacja warsztatu' ,1
   END
end
```

7.16 NoMoreDayBookingAfterPayment

Blokuje rezerwacje kolejnych dni konferencji po dokonaniu płatności.

```
create trigger NoMoreDayBookingAfterPayment on DayReservation after insert  as
begin
   SET NOCOUNT ON;
   IF EXISTS
   (
       select * from inserted as i inner join Payment as p on i.ConferenceReservationID
= p.ConferenceReservationID
       where p.PaymentDay is not null
   )
   begin
        ; THROW 50001 , 'Rezerwacja została już opłacona, nie można dokonywać kolejnych
rezerwacji miejsc na dzień. ' ,1
   end
end
```

7.17 NoMoreWorkshopBookingAfterPayment

Blokuje rezerwacje kolejnych warsztatów po dokonaniu płatności.

```
create trigger NoMoreWorkshopBookingAfterPayment on WorkshopsReservation after insert
as
begin
   SET NOCOUNT ON;
   IF EXISTS
   (
       select * from inserted as i inner join DayReservation as dr on
dr.DayReservationID = i.DayReservationID
       inner join Payment as p on dr.ConferenceReservationID = p.ConferenceReservationID
       where p.PaymentDay is not null
   )
   begin
        ; THROW 50001 , 'Rezerwacja została już opłacona, nie można dokonywać kolejnych
rezerwacji miejsc na warsztaty. ' ,1
   end
end
```

8. Generator Danych

8.1 Opis

Generator napisany został w języku Python. Generowany jest skrypt SQL wypełniający bazę danymi. Tworzone jest 100 firm i 72 konferencji - średnio trwają 2 dni, w każdym dniu są średnio 4 warsztaty. Jedna firma ma pomiędzy 1 a 100 pracowników.

8.2 Kod Generatora

```
import datetime
import random
import math
from faker import Faker
fake = Faker()


list_of_participants = list()
list_of_conferences = list()
list_of_company = list()
list_of_conferences_reservations = list()
list_of_days = list()
```

```
list_of_days_reservations = list()
list_of_days_participants = list()
list_of_prices_thresholds = list()
list_of_payments = list()
list_of_workshops = list()
list_of_workshop_reservations = list()
list_of_workshop_participants = list()


def conferences(n):
    cols = '(Name, beginDate, endDate)'
    q = 'INSERT INTO Conference ' + cols + ' VALUES '
    for i in range(0, n):
        start_date = datetime.date(2017, 1, 1) + datetime.timedelta(days=random.randint(0,
1095))
        end_date = start_date + datetime.timedelta(days=random.randint(1, 2))
        conference_name = fake.name()
        q = q + '(\'' + conference_name + '\', \'' \
        + str(start_date) + '\', \'' \
        + str(end_date) + '\')'
        list_of_conferences.append((i + 1, conference_name, start_date, end_date))
        if i < n - 1:
            q += ', '
    return q

def company(n):
    cols = '(CompanyName, Phone, [e-mail], Address)'
    q = 'INSERT INTO Company ' + cols + ' VALUES '
    for i in range(0, n):
        company_name = fake.name()
        phone = random.randint(111111111, 999999999)
        email = company_name + '@gmail.com'
        address = fake.address()
        q = q + '(\'' + company_name + '\', ' \
            + str(phone) + ', \'' \
            + email + '\', \'' \
            + str(address) + '\')'
        list_of_company.append((i + 1, company_name, phone, email, address))
        if i < n - 1:
            q += ', '
    return q

def conferences_reservations():
    cols = '(conferenceID, CompanyID, ReservationDate)'
    q = 'INSERT INTO ConferenceReservation ' + cols + ' VALUES '
    counter = 1
    for i, conference in enumerate(list_of_conferences):
```

```
        conference_id = conference[0]
        customers_number = random.randint(1, 6)
        customers_list = random.sample(range(1, len(list_of_company) + 1),
                                       min(customers_number, len(list_of_company)))
        for j, customerID in enumerate(customers_list):
            reservation_date = list_of_conferences[conference_id - 1][2] -
datetime.timedelta(days=random.randint(0, 60))
            q = q + '(' + str(conference_id) + ', ' \
            + str(customerID) + ', \'' \
            + str(reservation_date) + '\')'
            list_of_conferences_reservations.append((counter, conference_id, customerID,
reservation_date))
            counter += 1
            if j < len(customers_list) - 1:
                q += ', '
            elif i < len(list_of_conferences) - 1 and customers_list:
                q += ', '
    return q

def participants():
    cols = '(CompanyID, FirstName, LastName, StudentCard)'
    q_begin = 'INSERT INTO Participant ' + cols + ' VALUES '
    q = q_begin
    counter = 1
    for j, company in enumerate(list_of_company):
        company_id = company[0]
        employee_number = random.randint(1, 100)
        for i in range(0, employee_number):
            FirstName = fake.name()
            LastName = fake.name()
            possibbility = random.randint(1, 100)
            StudentCard = 'null'
            if possibbility > 70:
                StudentCard = random.randint(111111, 999999)
            q = q + '(' + str(company_id) + ', \'' + FirstName + '\', \'' \
                + LastName + '\', ' + str(StudentCard) + ')'
            list_of_participants.append((counter, company_id, FirstName, LastName,
StudentCard))
            counter += 1
            if counter % 1000 == 900:
                q += '\n' + q_begin
            elif i < employee_number - 1:
                q += ', '
            elif j < len(list_of_company) - 1:
                q += ', \n'
    return q
```

```
def days():
    cols = '(ConferenceID, DayDate, NumberOfPlaces)'
    q = 'INSERT INTO Days ' + cols + ' VALUES '
    counter = 1
    for i, conference in enumerate(list_of_conferences):
        conference_id = conference[0]
        number_of_days_in_conference = abs((conference[3] - conference[2]).days)
        for j in range(0, number_of_days_in_conference + 1):
            places = random.randint(50, 200)
            current_day = conference[2] + datetime.timedelta(days = j)
            q = q + '(' + str(conference_id) + ',\' ' \
            + str(current_day) + '\', ' \
            + str(places) + ')'
            list_of_days.append((counter, conference_id, conference[2] +
datetime.timedelta(days=j), places))
            counter += 1
            if j < number_of_days_in_conference or i < len(list_of_conferences) - 1:
                q += ', '
    return q

def days_reservation():
    cols = '(ConferenceReservationID, DayID, NumberOfPlaces)'
    q = 'INSERT INTO DayReservation ' + cols + ' VALUES '
    counter = 1
    for i, day in enumerate(list_of_days):
        day_id = day[0]
        conference_id = day[1]
        places = random.randint(1, day[3])
        booked = {conference_booking[0]: 0
                  for conference_booking in list_of_conferences_reservations
                  if conference_booking[1] == conference_id}
        bookings_for_conference = booked.keys()

        for j in range(0, places):
            booked[random.choice(tuple(bookings_for_conference))] += 1
        for j, bookingID in enumerate(booked.keys()):
            if booked[bookingID] > 0:
                q = q + '(' + str(bookingID) + ', ' \
                    + str(day_id) + ', ' + str(booked[bookingID]) + ')'
                list_of_days_reservations.append((counter, bookingID, day_id,
booked[bookingID]))
                counter += 1
                if j < len(booked) - 1:
                    q += ', '
                elif i < len(list_of_days) - 1:
                    q += ', '
    return q
```

```
def participant_for_day():
    cols = '(ParticipantID, DayReservationID)'
    q_begin = 'INSERT INTO ParticipantsForDay ' + cols + ' VALUES '
    q = q_begin
    counter = 1
    for i, day_reservation in enumerate(list_of_days_reservations):
        day_booking_id = day_reservation[0]
        participants_number = day_reservation[3]
        company_id = [reservation[2]
                      for reservation in list_of_conferences_reservations
                      if reservation[0] == day_reservation[1]]
        participants_list = [participant[0]
                             for participant in list_of_participants
                             if participant[1] == company_id[0]]
        participants_list2 = random.sample(participants_list,
                                           min(participants_number, len(participants_list)))
        for j, participantID in enumerate(participants_list2):
            q = q + '(' + str(participantID) + ', ' \
                + str(day_booking_id) + ')'
            list_of_days_participants.append((counter, participantID, day_booking_id))
            counter += 1
            if counter % 900 == 899:
                q += '\n' + q_begin
            elif j < len(participants_list2) - 1:
                q += ', '
            elif i < len(list_of_days_reservations) - 1 and participants_list2:
                q += ', '
    return q

def workshops():
    cols = '(DayID, Name, NumberOfPlaces, StartTime, EndTime, Price)'
    q = 'INSERT INTO Workshops ' + cols + ' VALUES '
    counter = 1
    for i, day in enumerate(list_of_days):
        day_id = day[0]
        workshops_number = random.randint(3, 5)
        for j in range(1, workshops_number):
            name = fake.name()
            places = int(math.sqrt(random.randint(1, day[3] * day[3])))
            start_time = datetime.datetime(20, 2, 2, 8, 0, 0) +
datetime.timedelta(hours=random.randint(0, 12))
            end_time = start_time + datetime.timedelta(minutes=random.randint(30, 180))
            start_time = start_time.time()
            end_time = end_time.time()
            price = random.randint(0, 5) * 10
            q = q + '(' + str(day_id) + ', \'' \
```

```
                + str(name) + '\', ' \
                + str(places) + ', \'' \
                + str(start_time) + '\', \'' \
                + str(end_time) + '\', ' \
                + str(price) + ')'
            list_of_workshops.append((counter, day_id, name, places, start_time, end_time,
price))
            counter += 1
            if j < workshops_number - 1 or i < len(list_of_days) - 1:
                q += ', '
    return q


def workshops_reservations():
    cols = '(DayReservationID, WorkshopID, NumberOfPlaces)'
    q_begin = 'INSERT INTO WorkshopsReservation ' + cols + ' VALUES '
    q = q_begin
    counter = 1
    for i, workshop in enumerate(list_of_workshops):
        workshop_id = workshop[0]
        day_id = workshop[1]
        if random.randint(0, 5) == 0:
            places = random.randint(1, workshop[3])
        else:
            places = workshop[3]
        booked_for_day = []
        for day_booking in list_of_days_reservations:
            if day_booking[2] == day_id:
                booked_for_day += [day_booking[0]] * day_booking[3]

        booked = random.sample(booked_for_day, min(places, len(booked_for_day)))
        booked_count = [(bookingID, booked.count(bookingID)) for bookingID in set(booked)]
        for j, dayBooking in enumerate(booked_count):
            q = q + '(' + str(dayBooking[0]) + ', ' \
                + str(workshop_id) + ', ' \
                + str(dayBooking[1]) + ')'
            list_of_workshop_reservations.append((counter, dayBooking[0], workshop_id,
dayBooking[1]))
            counter += 1
            if counter % 1000 == 900:
                q += '\n' + q_begin
            elif j < len(booked_count) - 1 or i < len(list_of_workshops) - 1:
                q += ', '
    return q

def workshops_participants():
    cols = '(ParticipantForDayID, WorkshopReservationID)'
```

```
    q_begin = 'INSERT INTO WorkshopsParticipants ' + cols + ' VALUES '
    q = q_begin
    counter = 1
    for i, workshops_reservation in enumerate(list_of_workshop_reservations):
        workshop_id = workshops_reservation[2]
        workshop_booking_id = workshops_reservation[0]
        day_booking_id = workshops_reservation[1]
        places = workshops_reservation[3] - 1
        start_time = list_of_workshops[workshop_id - 1][4]
        end_time = list_of_workshops[workshop_id - 1][5]
        day_participants = [dp[0] for dp in list_of_days_participants if dp[2] ==
day_booking_id]
        allowed_day_participants = set(day_participants)
        for day_participant_ID in day_participants:
            previous_workshops = [list_of_workshop_reservations[pe[2] - 1][2]
                                  for pe in list_of_workshop_participants if pe[1] ==
day_participant_ID]

            previous_times = [list_of_workshops[workshopID - 1][4:6] for workshopID in
previous_workshops]
            for time in previous_times:
                if end_time > time[0] and time[1] > start_time:
                    allowed_day_participants.remove(day_participant_ID)
                    break
        if random.randint(0, 20) == 0:
            number = random.randint(0, len(allowed_day_participants))
        else:
            number = len(allowed_day_participants)
        booked = random.sample(allowed_day_participants, min(places, number))
        for j, day_participant_ID in enumerate(booked):
            q = q + '(' + str(day_participant_ID) + ', ' \
                + str(workshop_booking_id) + ')'
            list_of_workshop_participants.append((counter, day_participant_ID,
workshop_booking_id))
            counter += 1
            if counter % 1000 == 900:
                q += '\n' + q_begin
            elif j < len(booked) - 1 or i < len(list_of_workshop_reservations) - 1:
                q += ', '
    return q


def payments():
    cols = '(ConferenceReservationID, PaymentDay)'
    q_begin = 'INSERT INTO Payment ' + cols + ' VALUES '
    q = q_begin
    for i, booking in enumerate(list_of_conferences_reservations):
```

```
        conference_booking_id = booking[0]
        conference_id = booking[1]
        register_date = booking[3]
        conference = list_of_conferences[conference_id - 1]
        start_date = conference[2]
        day_diff = (start_date - register_date).days
        if day_diff > 7:
            day_diff = 7
        payment_date = register_date + datetime.timedelta(days=random.randint(0, day_diff))
        q = q + '(' + str(conference_booking_id) + ', \'' \
            + str(payment_date) + '\')'
        list_of_payments.append((i + 1, conference_booking_id, payment_date))
        if i % 1000 == 900:
            q += '\n' + q_begin
        elif i < len(list_of_conferences_reservations) - 1:
            q += ', '
    return q


def prices():
    cols = '(ConferenceID, beginDate, Price)'
    q = 'INSERT INTO PriceThreshold ' + cols + ' VALUES '
    for i, conference in enumerate(list_of_conferences):
        number_of_thresholds = random.randint(2, 5)
        price = random.randint(10, 100)
        date = conference[2]
        current_date = date - datetime.timedelta(days=14*(number_of_thresholds + 1))
        for j in range(0, number_of_thresholds):
            current_date = current_date + datetime.timedelta(days=14)
            new_price = price + j* 20
            q = q + '(' + str(conference[0]) + ', \'' \
                + str(current_date) + '\', ' \
                + str(new_price) + ')'
            list_of_prices_thresholds.append((conference[0], current_date, new_price))
            if j < number_of_thresholds - 1:
                q += ', '
            elif i < len(list_of_conferences) - 1:
                q += ', '
    return q



print(company(100) + "\n" +
      conferences(72) + "\n" +
      participants() + "\n" +
      days() + "\n" +
      conferences_reservations() + "\n" +
```

```
    days_reservation() + "\n" +
    participant_for_day() + "\n" +
    prices() + "\n" +
    workshops() + "\n" +
    workshops_reservations() + "\n" +
    workshops_participants() + "\n" +
    payments() + "\n")
```