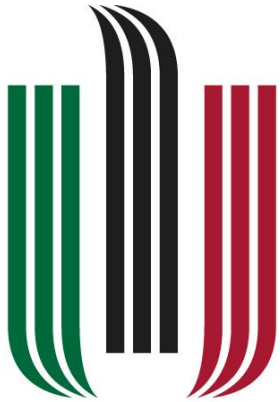


AGH

AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

Podstawy baz danych
Projekt i implementacja systemu bazodanowego

System zarządzania konferencjami

Paulina Kowalczyk, Julia Plewa

WIEiT

Kraków, styczeń 2018

1. Opis użytkowników

1.1. Klienci

Klientami mogą być zarówno osoby indywidualne jak i firmy, którzy rejestrują się na konferencje za pomocą systemu WWW. Przy rejestracji firma nie musi podawać od razu listy uczestników - może zarezerwować odpowiednią ilość miejsc na określone dni oraz na warsztaty, natomiast na 2 tygodnie przed rozpoczęciem musi te dane uzupełnić.

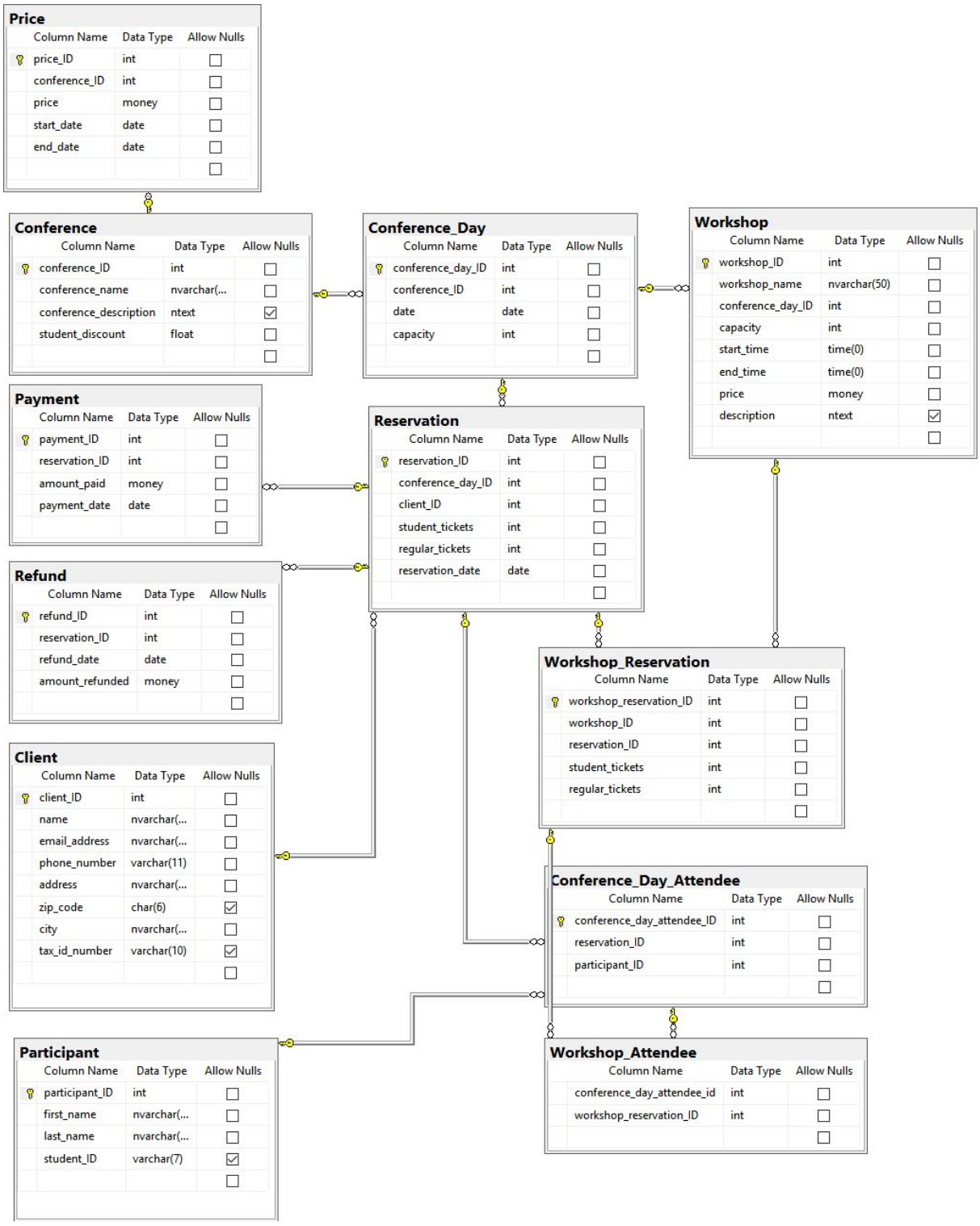
1.2. Uczestnicy

Uczestnikami konferencji są osoby. Dla konferencji kilkudniowych, uczestnicy mogą rejestrować się na dowolne z tych dni. Mogą także dokonywać rezerwacji na warsztaty - muszą być jednak zarejestrowani tego dnia na konferencję, aby móc w nich uczestniczyć. Kilka warsztatów może trwać równocześnie, ale uczestnik nie może zarejestrować się na więcej niż jeden warsztat, który trwa w tym samym czasie.

1.3. Pracownicy firmy

Pracownicy odpowiedzialni są za kontakt z firmą, która dokonała rezerwacji miejsc na konferencje, ale na 2 tygodnie przed rozpoczęciem nie uzupełniła jeszcze danych o uczestnikach.

2. Schemat bazy danych



Rys. 1. Schemat bazy danych

3. Opisy tabel

3.1. Client

Klientem mogą być osoby prywatne oraz firmy. Przechowywane są informacje takie jak client_ID będący kluczem glownym, dane kontaktowe (adres e-mail, numer telefonu), dane adresowe (ulica, kod pocztowy, miasto) oraz numer NIP (tax_id_number), gdzie dopuszczalna jest wartość NULL. Wartość NULL oznacza, że mamy do czynienia z klientem indywidualnym, a uzupełniony NIP wskazuje na firmę. Poprawność adresu email, kodu pocztowego i numeru NIP jest sprawdzana przy pomocy pomocy warunków integralności.

```
IF OBJECT_ID('dbo.Client', 'U') IS NOT NULL
  DROP TABLE dbo.Client

CREATE TABLE [dbo].[Client](
  [client_ID] [int] NOT NULL primary key identity (1,1),
  [name] [nvarchar](50) NOT NULL,
  [email_address] [nvarchar](50) UNIQUE NOT NULL,
  [phone_number] [varchar](11) NOT NULL,
  [address] [nvarchar](50) NOT NULL,
  [zip_code] [char](6) NOT NULL,
  [city] [nvarchar](50) NOT NULL,
  [tax_id_number] [varchar](10) NULL,

  CONSTRAINT check_zip_code CHECK (zip_code LIKE
  '[0-9][0-9]-[0-9][0-9][0-9]'),
  CONSTRAINT check_email_address CHECK (email_address LIKE '%_@_%._%'),
  CONSTRAINT check_tax_id_number CHECK (tax_id_number LIKE
  '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]')
)
```

3.2. Conference

Tabela zawiera informacje o poszczególnych konferencjach: conference_ID (klucz główny), nazwę konferencji, opcjonalny opis oraz informację o zniżce dla studentów (stałej w obrębie całej konferencji). Wysokość zniżki studenckiej musi mieścić się w zakresie od 0 do 1.

```
IF OBJECT_ID('dbo.Conference', 'U') IS NOT NULL
  DROP TABLE dbo.Conference

CREATE TABLE [dbo].[Conference](
  [conference_ID] [int] NOT NULL primary key identity (1,1),
  [conference_name] [nvarchar](50) NOT NULL,
  [conference_description] [ntext] NULL,
  [student_discount] [float] NULL,

  CONSTRAINT check_student_discount CHECK (student_discount >= 0 and
  student_discount <= 1)
)
```

3.3. Conference_Day

Tabela zawiera informacje o poszczególnych dniach konferencji: klucz główny conference_day_ID, klucz obcy conference_ID, który określa konferencję, do której przynależy dany dzień, datę oraz łączną liczbę biletów udostępnionych na ten dzień. Poprawność liczby miejsc jest sprawdzana - nie może być ujemna.

```
IF OBJECT_ID('dbo.Conference_Day', 'U') IS NOT NULL
  DROP TABLE dbo.Conference_Day

CREATE TABLE [dbo].[Conference_Day](
  [conference_day_ID] [int] NOT NULL primary key IDENTITY(1,1),
  [conference_ID] [int] NOT NULL foreign key references Conference
  (conference_ID),
  [date] [date] NOT NULL,
  [capacity] [int] NOT NULL,

  CONSTRAINT check_capacity CHECK (capacity >= 0)
)
```

3.4. Conference_Day_Attendee

Tabela zawiera informacje o uczestnikach konferencji w danym dniu: klucz główny conference_day_attendee_ID oraz klucze obce reservation_ID i participant_ID, które pozwalają na połączenie konkretnego uczestnika z konkretną rezerwaccją.

```
IF OBJECT_ID('dbo.Conference_Day_Attendee', 'U') IS NOT NULL
  DROP TABLE dbo.Conference_Day_Attendee

CREATE TABLE [dbo].[Conference_Day_Attendee](
  [conference_day_attendee_ID] [int] NOT NULL primary key identity (1,1),
  [reservation_ID] [int] NOT NULL foreign key references Reservation
  (reservation_ID),
  [participant_ID] [int] NOT NULL foreign key references Participant
  (participant_ID)
)
```

3.5. Participant

Tabela zawiera informacje o uczestnikach konferencji. Podawane są one przez klientów rejestrujących swoich uczestników lub telefonicznie przez samych uczestników. Każdy uczestnik posiada klucz główny participant_ID oraz imię i nazwisko. Opcjonalnie w celu uzyskania zniżki podany może zostać numer legitymacji studenckiej.

```
IF OBJECT_ID('dbo.Participant', 'U') IS NOT NULL
  DROP TABLE dbo.Participant

CREATE TABLE [dbo].[Participant](
  [participant_ID] [int] NOT NULL primary key identity(1,1),
  [first_name] [nvarchar](50) NOT NULL,
  [last_name] [nvarchar](50) NOT NULL,
  [student_ID] [varchar(7)] NULL
)
```

3.6. Payment

Tabela zawiera informacje o płatnościach: payment_ID będący kluczem głównym, klucz obcy reservation_ID oraz zapłaconą kwotę i datę zarejestrowania płatności w systemie. Opłacona kwota musi być większa od zera, a data płatności nie może znajdować się w przyszłości. W tabeli rejestrowane są płatności za warsztaty oraz za dni konferencji.

```
IF OBJECT_ID('dbo.Payment', 'U') IS NOT NULL
  DROP TABLE dbo.Payment

CREATE TABLE [dbo].[Payment](
  [payment_ID] [int] NOT NULL primary key identity(1,1),
  [reservation_ID] [int] NOT NULL foreign key references Reservation
  (reservation_ID),
  [amount_paid] [money] NOT NULL CHECK (amount_paid > 0),
  [payment_date] [date] NOT NULL CHECK (payment_date <= GETDATE())
)
```

3.7. Price

Tabela przechowuje informacje odnośnie progów cenowych konferencji. Zawiera klucz główny price_ID, klucz obcy conference_ID, nieujemną cenę typu money oraz daty start_date i end_date określające przedział czasu, kiedy obowiązuje dana cena.

```
IF OBJECT_ID('dbo.Price', 'U') IS NOT NULL
  DROP TABLE dbo.Price

CREATE TABLE [dbo].[Price](
  [price_ID] [int] NOT NULL primary key identity (1,1),
  [conference_ID] [int] NOT NULL foreign key references conference
  (conference_ID),
  [price] [money] NOT NULL check (price >= 0),
  [start_date] [date] NOT NULL,
  [end_date] [date] NOT NULL
)
```

3.8. Refund

Tabela przechowuje informacje odnośnie zwrotów płatności w przypadku anulowanych rezerwacji. Zawiera klucz główny refund_ID, klucz obcy reservation_ID określający rezerwację, która została anulowana, datę refund_date oraz kwotę amount_refunded. Pole amount_refunded zawiera informację o zwróconej kwocie i pomaga kontrolować przepływ pieniędzy w firmie. Data refund_date nie może znajdować się w przyszłości.

```
IF OBJECT_ID('dbo.Refund', 'U') IS NOT NULL
  DROP TABLE dbo.Refund

CREATE TABLE [dbo].[Refund](
  [refund_ID] [int] NOT NULL primary key identity (1,1),
  [reservation_ID] [int] NOT NULL foreign key references Reservation
  (reservation_ID),
  [refund_date] [date] NOT NULL check (refund_date <= GETDATE()),
  [amount_refunded] [money] NOT NULL check (amount_refunded >= 0)
)
```

3.9. Reservation

Tabela przechowuje informacje odnośnie rezerwacji dokonanych przez klientów na poszczególne dni konferencji. Zawiera klucz pierwotny reservation_ID, klucz obcy conference_day_ID klucz obcy client_ID, nieujemne kolumny student_tickets i regular_tickets określające liczbę poszczególnych biletów w rezerwacji oraz reservation_date określające datę rezerwacji. Na podstawie daty rezerwacji określana jest odpowiednia cena z tabeli Prices. Od tego dnia klient ma tydzień na opłacenie rezerwacji - w przeciwnym wypadku zostaje ona anulowana.

```
IF OBJECT_ID('dbo.Reservation', 'U') IS NOT NULL
  DROP TABLE dbo.Reservation

CREATE TABLE [dbo].[Reservation](
  [reservation_ID] [int] NOT NULL primary key identity (1,1),
  [conference_day_ID] [int] NOT NULL foreign key references Conference_Day
  (conference_day_ID),
  [client_ID] [int] NOT NULL foreign key references Client(client_ID),
  [student_tickets] [int] NOT NULL check (student_tickets >= 0),
  [regular_tickets] [int] NOT NULL check (regular_tickets >= 0),
  [reservation_date] [date] NOT NULL
)
```

3.10. Workshop

Tabela przechowuje informacje odnośnie warsztatów odbywających się w danym dniu konferencji. Zawiera klucz pierwotny workshop_ID, klucz obcy conference_day_ID określający dzień konferencji, nieujemeną kolumnę capacity określająca dopuszczalną liczbę uczestników, start_time i end_time typu time(0) (hh:mm:ss) określające godzinę rozpoczęcia i zakończenia warsztatów, nieujemną kolumnę price określającą cenę warsztatów oraz opcjonalną kolumnę description typu ntext z opisem warsztatów.

```
IF OBJECT_ID('dbo.Workshop', 'U') IS NOT NULL
  DROP TABLE dbo.Workshop

CREATE TABLE [dbo].[Workshop](
  [workshop_ID] [int] NOT NULL primary key identity (1,1),
  [conference_day_ID] [int] NOT NULL foreign key references Conference_Day
  (conference_day_ID),
  [capacity] [int] NOT NULL check (capacity >= 0),
  [start_time] [time](0) NOT NULL,
  [end_time] [time](0) NOT NULL,
  [price] [money] NOT NULL check (price >= 0),
  [description] [ntext] NULL
)
```

3.11. Workshop_Attendee

Tabela przechowuje informacje odnośnie uczestników warsztatów. Zawiera klucz obcy conference_day_attendee_ID (tabela Conference_Day_Attendee łączy uczestnika konferencji z danym dniem konferencji) oraz klucz obcy workshop_reservation_ID, który łączy tego uczestnika z rezerwacją na konkretny warsztat.

```
IF OBJECT_ID('dbo.Workshop_Attendee', 'U') IS NOT NULL
  DROP TABLE dbo.Workshop_Attendee

CREATE TABLE [dbo].[Workshop_Attendee](
  [conference_day_attendee_ID] [int] NOT NULL foreign key references
  Conference_Day_Attendee (conference_day_attendee_ID),
  [workshop_reservation_ID] [int] NOT NULL foreign key references
  Workshop_Reservation (workshop_reservation_ID)
)
```

3.12. Workshop_Reservation

Tabela przechowuje informacje odnośnie rezerwacji na warsztaty. Posiada klucz główny workshop_reservation_ID, klucz obcy workshop_ID określający warsztat, klucz obcy reservation_ID łączący tę rezerwację z rezerwacją na dzień konferencji oraz nieujemne wartości regular_tickets i student_tickets określające liczbę biletów zarezerwowanych na tym warsztacie.

```
IF OBJECT_ID('dbo.Workshop_Reservation', 'U') IS NOT NULL
  DROP TABLE dbo.Workshop_Reservation

CREATE TABLE [dbo].[Workshop_Reservation](
  [workshop_reservation_ID] [int] NOT NULL primary key identity (1,1),
  [workshop_ID] [int] NOT NULL foreign key references Workshop (workshop_ID),
  [reservation_ID] [int] NOT NULL foreign key references Reservation
  (reservation_ID),
  [regular_tickets] [int] NOT NULL check (regular_tickets >= 0),
  [student_tickets] [int] NOT NULL check (student_tickets >= 0)
)
```

4. Procedury

4.1. Dodawanie konferencji

```
CREATE PROCEDURE [dbo].[add_conference](
  @conference_name nvarchar(50),
  @conference_description ntext,
  @student_discount float
) AS
BEGIN
  SET NOCOUNT ON;
  INSERT INTO Conference (conference_name, conference_description,
  student_discount)
  VALUES (@conference_name, @conference_description, @student_discount)
END
```

4.2. Dodawanie dni konferencji

```
CREATE PROCEDURE [dbo].[add_conference_day](
  @conference_id int,
  @date date,
  @capacity int
) AS
BEGIN
  SET NOCOUNT ON;
  IF EXISTS (SELECT * FROM Conference_Day WHERE conference_ID = @conference_ID
  and date = @date)
    BEGIN
      RAISERROR ('This date has already been added to the conference',14,1)
      RETURN
    END
  INSERT INTO Conference_Day (conference_ID, date, capacity)
  VALUES (@conference_ID, @date, @capacity)
END
```

4.3. Dodawanie warsztatów

```
CREATE PROCEDURE [dbo].[add_workshop](
  @workshop_name nvarchar(50),
  @conference_day_id int,
  @capacity int,
  @start_time time(0),
  @end_time time (0),
  @price money,
  @description ntext
) AS
BEGIN
  SET NOCOUNT ON;
  IF @capacity > (SELECT cd.capacity FROM conference_day AS cd WHERE
  cd.conference_day_id = @conference_day_id)
    BEGIN
      RAISERROR ('Workshop''s capacity exceeds day''s capacity',14,1)
      RETURN
    END
  IF DATEDIFF(mi, @start_time, @end_time) <= 0
    BEGIN
      RAISERROR ('Incorrect start_time and end_time',14,1)
      RETURN
    END
  INSERT INTO Workshop (workshop_name, conference_day_id, capacity,
  start_time, end_time, price, description)
  VALUES (@workshop_name, @conference_day_id, @capacity, @start_time,
  @end_time, @price, @description)
END
```

4.4. Dodawanie klientów

```
CREATE PROCEDURE [dbo].[add_client](
  @name nvarchar(50),
  @email_address nvarchar(50),
  @phone_number varchar(11),
  @address nvarchar(50),
  @zip_code char(6),
  @city nvarchar(50),
  @tax_id_number varchar(10)
) AS BEGIN
  SET NOCOUNT ON;
  INSERT INTO Client(name, email_address, phone_number, address, zip_code,
  city, tax_id_number)
  VALUES (@name, @email_address, @phone_number, @address, @zip_code, @city,
  @tax_id_number)
END
```

4.5. Dodawanie uczestników

```
CREATE PROCEDURE [dbo].[add_participant](
  @first_name nvarchar(50),
  @last_name nvarchar(50),
  @student_id varchar(7)
) AS
BEGIN
  SET NOCOUNT ON;
  INSERT INTO Participant(first_name, last_name, student_ID)
  VALUES (@first_name, @last_name, @student_ID)
END
```

4.6. Dodawanie płatności

```
CREATE PROCEDURE [dbo].[add_payment](
  @reservation_ID INT,
  @amount_paid MONEY
) AS
BEGIN
  SET NOCOUNT ON
  INSERT INTO Payment (reservation_ID, amount_paid, payment_date)
  VALUES (@reservation_ID, @amount_paid, GETDATE())
END
```

4.7. Dodawanie progów cenowych

```
CREATE PROCEDURE [dbo].[add_price](
  @conference_ID int,
  @price int,
  @start_date date,
  @end_date date
) AS
BEGIN
  IF (@start_date > @end_date or @start_date < GETDATE() or @end_date <
  GETDATE())
    BEGIN
      RAISERROR ('Incorrect dates',14,1)
      RETURN
    END
  SET NOCOUNT ON
  INSERT INTO Price (conference_ID, price, start_date, end_date)
  VALUES (@Conference_ID,@Price,@Start_date, @End_date)
END
```

4.8. Zapisywanie uczestnika na dzień konferencji

```
CREATE PROCEDURE [dbo].[add_conference_day_attendee] (
  @participant_id int,
  @reservation_id varchar(50)
) AS
BEGIN
  SET NOCOUNT ON;
  INSERT INTO Conference_Day_Attendee (participant_ID, reservation_ID)
  VALUES (@participant_ID, @reservation_ID)
END
```

4.9. Zapisywanie uczestnika na warsztat

```
CREATE PROCEDURE [dbo].[add_workshop_attendee] (
  @conference_day_attendee_ID int,
  @workshop_reservation_id int
) AS
BEGIN
  SET NOCOUNT ON
  INSERT dbo.Workshop_Attendee (conference_day_attendee_id,
  workshop_reservation_ID)
  VALUES (@conference_day_attendee_ID, @workshop_reservation_ID)
END
```

4.10. Rezerwacja miejsc na dany dzień konferencji

```
CREATE PROCEDURE [dbo].[add_conference_day_reservation](
  @conference_day_id int,
  @client_ID int,
  @student_tickets int,
  @regular_tickets int
) AS
BEGIN
  SET NOCOUNT ON;
  INSERT INTO Reservation (conference_day_ID, client_ID, student_tickets,
  regular_tickets, reservation_date)
  VALUES (@conference_day_ID, @client_ID, @student_tickets, @regular_tickets,
  GETDATE())
END
```

4.11. Rezerwacja miejsc na dany warsztat

```
CREATE PROCEDURE [dbo].[add_workshop_reservation](
  @workshop_id int,
  @reservation_id int,
  @student_tickets int,
  @normal_tickets int
) AS
BEGIN
  SET NOCOUNT ON;
  INSERT INTO Workshop_Reservation (workshop_ID, reservation_ID,
  student_tickets, normal_tickets)
  VALUES (@workshop_ID, @reservation_ID, @student_tickets, @normal_tickets)
END
```

4.12. Zmiana liczby miejsc dla danego dnia konferencji

```
CREATE PROCEDURE [dbo].[change_conference_day_capacity](
  @conference_day_ID int,
  @capacity int
)AS
BEGIN
  SET NOCOUNT ON
  IF ((SELECT sum(student_tickets + regular_tickets) FROM Reservation WHERE
  conference_day_ID = @conference_day_ID) > @capacity)
    BEGIN
      RAISERROR ('Number of reserved tickets exceeds the requested capacity',
      14,1)
      RETURN
    END
  UPDATE dbo.Conference_Day
  SET capacity = @capacity
  WHERE conference_day_ID = @Conference_day_ID
END
```

4.13. Zmiana liczby miejsc dla danego warsztatu

```
CREATE PROCEDURE [dbo].[change_workshop_capacity](
  @workshop_ID int,
  @capacity int
) AS BEGIN
  SET NOCOUNT ON
  IF ((SELECT sum(student_tickets + regular_tickets) FROM Workshop_Reservation
  WHERE workshop_ID = @workshop_ID) > @capacity)
    BEGIN
      RAISERROR ('Number of reserved tickets exceeds the requested capacity',
      14,1)
      RETURN
    END
  UPDATE dbo.Workshop
  SET capacity = @capacity
  WHERE workshop_ID = @workshop_ID
END
```

4.14. Anulowanie danej rezerwacji na dzień wraz z warsztatami

```
CREATE PROCEDURE [dbo].[cancel_reservation](
  @reservation_id int
) AS
BEGIN
  IF @reservation_id IS NULL
    BEGIN
      RAISERROR ('Null values are not allowed', 14,1)
      RETURN
    END
  IF EXISTS (SELECT * FROM Refund where reservation_id = @reservation_id)
    BEGIN
      RAISERROR ('This reservation had already been cancelled', 14,1)
      RETURN
    END
  DELETE FROM Workshop_Attendee WHERE conference_day_attendee_id IN (SELECT
  conference_day_attendee_id FROM Conference_Day_Attendee WHERE reservation_ID
  = @reservation_id)
  DELETE FROM Conference_Day_Attendee WHERE reservation_ID = @reservation_id

  DECLARE @amount_refunded MONEY = COALESCE((SELECT sum(amount_paid) FROM
  dbo.Payment WHERE reservation_ID = @reservation_id),0);

  INSERT INTO Refund (reservation_ID, refund_date, amount_refunded)
  VALUES (@reservation_id, GETDATE(), @amount_refunded)
END
```

4.15. Zmiana liczby zarezerwowanych miejsc dla warsztatu

```
CREATE PROCEDURE [dbo].[add_workshop_tickets_to_reservation](
  @workshop_reservation_id int,
  @student_tickets int,
  @regular_tickets int
) AS
BEGIN
  SET NOCOUNT ON

  DECLARE @reservation_id int = (SELECT reservation_id FROM
  Workshop_Reservation WHERE workshop_reservation_ID =
  @workshop_reservation_id)

  IF EXISTS (SELECT * FROM Refund WHERE reservation_ID = @reservation_id)
    BEGIN
      RAISERROR ('Reservation had been cancelled', 14,1)
      RETURN
    END

  DECLARE @conference_id int = (SELECT conference_ID FROM Conference_Day WHERE
  conference_day_ID = (SELECT conference_day_ID FROM Reservation WHERE
  reservation_ID = @reservation_id));

  IF (DATEDIFF(dd, GETDATE(), (SELECT min(date) FROM Conference_Day WHERE
  conference_ID = @conference_id)) <= 7)
    BEGIN
      RAISERROR ('Conference is less than a week ahead', 14,1)
      RETURN
    END

  DECLARE @student_day_tickets int = (SELECT student_tickets FROM Reservation
  WHERE reservation_ID = @reservation_id);
  DECLARE @regular_day_tickets int = (SELECT @regular_tickets FROM Reservation
  WHERE reservation_ID = @reservation_id);

  IF ((((SELECT student_tickets FROM Workshop_Reservation WHERE
  workshop_reservation_ID = @workshop_reservation_id) + @student_tickets) >
  @student_day_tickets) or
  (((SELECT regular_tickets FROM Workshop_Reservation WHERE
  workshop_reservation_ID = @workshop_reservation_id) + @regular_tickets) >
  @regular_day_tickets))
    BEGIN
      RAISERROR ('Requested number of tickets exceeds the number of purchased
      day tickets', 14,1)
      RETURN
    END

  DECLARE @workshop_id int = (SELECT workshop_ID FROM Workshop_Reservation
```

```
  WHERE workshop_reservation_ID = @workshop_reservation_id);
  DECLARE @capacity int = (SELECT capacity FROM Workshop WHERE workshop_ID =
  @workshop_id);

  IF ((@capacity - (SELECT sum(student_tickets + regular_tickets) FROM
  Workshop_Reservation WHERE workshop_ID = @workshop_id)) < (@student_tickets
  +@regular_tickets))
    BEGIN
      RAISERROR ('Requested number of tickets is not available', 14,1)
      RETURN
    END

  UPDATE Workshop_Reservation SET
  student_tickets = ((SELECT student_tickets FROM Workshop_Reservation WHERE
  workshop_reservation_ID = @workshop_reservation_id) + @student_tickets),
  regular_tickets = ((SELECT regular_tickets FROM Workshop_Reservation WHERE
  workshop_reservation_ID = @workshop_reservation_id) + @regular_tickets)
  WHERE reservation_ID = @reservation_id
END
```

4.16. Zmiana liczby zarezerwowanych miejsc dla dnia konferencji

```
CREATE PROCEDURE [dbo].[add_day_tickets_to_reservation](
  @reservation_id int,
  @student_tickets int,
  @regular_tickets int
) AS
BEGIN
  SET NOCOUNT ON
  IF EXISTS (SELECT * FROM Refund WHERE reservation_ID = @reservation_id)
    BEGIN
      RAISERROR ('Reservation had been cancelled', 14,1)
      RETURN
    END
  DECLARE @conference_id int = (SELECT conference_ID FROM Conference_Day WHERE
  conference_day_ID = (SELECT conference_day_ID FROM Reservation WHERE
  reservation_ID = @reservation_id));
  IF (DATEDIFF(dd, GETDATE(), (SELECT min(date) FROM Conference_Day WHERE
  conference_ID = @conference_id)) <= 7)
    BEGIN
      RAISERROR ('Conference is less than a week ahead', 14,1)
      RETURN
    END
  DECLARE @capacity int = (SELECT capacity FROM Conference_Day WHERE
  conference_day_ID = (SELECT conference_day_ID FROM Reservation WHERE
  reservation_ID = @reservation_id));
  IF ((@capacity - (SELECT sum(student_tickets + regular_tickets) FROM
  Reservation)) < (@student_tickets +@regular_tickets) )
    BEGIN
      RAISERROR ('Requested number of tickets is not available', 14,1)
      RETURN
    END
  UPDATE dbo.Reservation SET
    student_tickets = ((SELECT student_tickets FROM Reservation WHERE
    reservation_ID = @reservation_id) + @student_tickets),
    regular_tickets = ((SELECT regular_tickets FROM Reservation WHERE
    reservation_ID = @reservation_id) + @regular_tickets)
    WHERE reservation_ID = @reservation_id
END
```

4.17. Usunięcie uczestnika z rezerwacji na dzien

```
CREATE PROCEDURE [dbo].[remove_day_attendee]
  @participant_ID int,
  @reservation_ID int
) AS
BEGIN
  IF @participant_ID IS NULL or @reservation_ID IS NULL
    BEGIN
      RAISERROR ('Null values are not allowed here', 14,1)
      RETURN
    END

  DECLARE @conference_day_attendee_id int = (SELECT conference_day_attendee_id
  FROM Conference_Day_Attendee
  WHERE reservation_ID = @reservation_ID and participant_ID =
  @participant_ID);
  DECLARE @conference_id int = (SELECT conference_ID FROM Conference_Day WHERE
  conference_day_ID = (SELECT conference_day_ID FROM Reservation WHERE
  reservation_ID = @reservation_ID))

  IF (DATEDIFF(dd, GETDATE(), (SELECT min(date) FROM Conference_Day WHERE
  conference_ID = @conference_id)) <= 14)
    BEGIN
      RAISERROR ('Conference is less than two weeks ahead', 14,1)
      RETURN
    END

  DELETE FROM Workshop_Attendee
  WHERE conference_day_attendee_id = @conference_day_attendee_id
  AND workshop_reservation_ID in (SELECT workshop_reservation_ID FROM
  Workshop_Reservation WHERE reservation_ID = @reservation_ID)

  DELETE FROM Conference_Day_Attendee
  WHERE conference_day_attendee_id = @conference_day_attendee_id
  AND participant_ID = @participant_ID

END
```

4.18. Usunięcie uczestnika z rezerwacji na warsztaty

```
CREATE PROCEDURE [dbo].[remove_workshop_attendee](
  @participant_ID INT,
  @workshop_reservation_ID int
) AS
BEGIN
  IF @participant_ID IS NULL or @workshop_reservation_ID IS NULL
    BEGIN
      RAISERROR ('Null values are not allowed here', 14,1)
      RETURN
    END

  DECLARE @conference_day_attendee_id int = (SELECT
  wa.conference_day_attendee_id FROM Workshop_Attendee AS wa JOIN
  Conference_Day_Attendee AS cda ON cda.conference_day_attendee_ID =
  wa.conference_day_attendee_id WHERE cda.participant_ID=@participant_ID)
  DECLARE @conference_id int = (SELECT conference_ID FROM Conference_Day WHERE
  conference_day_ID = (SELECT conference_day_ID FROM Reservation WHERE
  reservation_ID = (SELECT reservation_ID FROM Conference_Day_Attendee WHERE
  conference_day_attendee_ID = @conference_day_attendee_id)));

  IF (DATEDIFF(dd, GETDATE(), (SELECT min(date) FROM Conference_Day WHERE
  conference_ID = @conference_id)) <= 14)
    BEGIN
      RAISERROR ('Conference is less than two weeks ahead', 14,1)
      RETURN
    END

  DELETE FROM dbo.Workshop_Attendee
  WHERE conference_day_attendee_id = @conference_day_attendee_id AND
  workshop_reservation_ID = @workshop_reservation_id
END
```

4.19. Anulowanie rezerwacji nieopłaconych przez tydzień

```
CREATE procedure [dbo].[cancel_unpaid_reservations]
AS
BEGIN
  DECLARE @reservation_ID int
  WHILE EXISTS(SELECT * FROM Reservation AS r WHERE (dbo.reservation_price
  (r.reservation_ID) > (SELECT sum(p.amount_paid) FROM Payment AS p WHERE
  p.reservation_ID = r.reservation_ID)) and (DATEDIFF(dd, r.reservaation_date,
  GETDATE()) >= 7))
    BEGIN
      SELECT TOP 1 @reservation_ID = reservation_ID FROM Reservation AS r
      WHERE (dbo.reservation_price (r.reservation_ID) > (SELECT
      sum(p.amount_paid) FROM Payment AS p WHERE r.reservation_ID =
      p.reservation_ID)) and (DATEDIFF(dd, r.reservation_date, GETDATE()) >=
      7)

      EXEC dbo.cancel_reservation @reservation_ID
    END
END
```

5. Procedury wykorzystane do generowania danych

5.1. Dodawanie płatności z przeszłości

```
CREATE PROCEDURE [dbo].[gen_add_payment](
  @reservation_ID INT ,
  @amount_paid MONEY,
  @payment_date date
) AS
BEGIN
  SET NOCOUNT ON
  INSERT INTO Payment (reservation_ID, amount_paid, payment_date)
  VALUES (@reservation_ID, @amount_paid, @payment_date)
END
```

5.2. Dodawanie rezerwacji z przeszłości

```
CREATE PROCEDURE [dbo].[gen_add_reservation](
  @conference_day_id INT,
  @client_ID INT,
  @student_tickets INT,
  @regular_tickets INT,
  @reservation_date date
) AS
BEGIN
  SET NOCOUNT ON;
  INSERT INTO Reservation (conference_day_ID, client_ID, student_tickets,
  regular_tickets, reservation_date)
  VALUES (@conference_day_id, @client_ID, @student_tickets, @regular_tickets,
  @reservation_date)
END
```

5.3. Anulowanie rezerwacji z przeszłości

```
CREATE PROCEDURE [dbo].[gen_cancel_reservation](
  @reservation_id int
) AS
BEGIN
  IF @reservation_id IS NULL
    BEGIN
      RAISERROR ('Null values are not allowed', 14,1)
      RETURN
    END

  DELETE FROM Workshop_Attendee WHERE conference_day_attendee_id in (SELECT
  conference_day_attendee_id FROM Conference_Day_Attendee WHERE reservation_ID
  = @reservation_id)
  DELETE FROM Conference_Day_Attendee WHERE reservation_ID = @reservation_id

  DECLARE @amount_refunded MONEY = COALESCE((SELECT sum(amount_paid) FROM
  dbo.Payment WHERE reservation_ID = @reservation_id),0);

  INSERT INTO Refund (reservation_ID, refund_date, amount_refunded)
  VALUES (@reservation_id, (dateadd(dd, 7, (SELECT reservation_date FROM
  Reservation WHERE reservation_ID = @reservation_id))), @amount_refunded)
END
```

5.4. Dodawanie progów cenowych z przeszłości

```
CREATE PROCEDURE [dbo].[gen_add_price](
  @conference_ID int,
  @price int,
  @start_date date,
  @end_date date
) AS
BEGIN
  IF (@start_date > @end_date)
    BEGIN
      RAISERROR ('Incorrect dates',14,1)
      RETURN
    END
  SET NOCOUNT ON
  INSERT INTO Price (conference_ID, price, start_date, end_date)
  VALUES (@Conference_ID,@Price,@Start_date, @End_date)
END
```

6. Triggery

6.1. Weryfikacja dostępnej liczby miejsc na dzień konferencji

```
CREATE TRIGGER [dbo].[check_if_day_reservation_exceeds_capacity]
  ON [dbo].[Reservation]
  AFTER INSERT, UPDATE
AS
BEGIN
  SET NOCOUNT ON;

  DECLARE @conference_day_ID int = (SELECT conference_day_ID FROM inserted);
  DECLARE @reserved_tickets int = (SELECT sum(student_tickets +
  regular_tickets) FROM Reservation WHERE conference_day_ID =
  @conference_day_ID and NOT EXISTS (SELECT * FROM Refund AS ref WHERE
  ref.reservation_ID = reservation_ID))

  IF (@reserved_tickets > (SELECT capacity FROM Conference_Day WHERE
  conference_day_ID = @conference_day_id))
    BEGIN;
      THROW 51000, 'Requested number of tickets exceeds day''s capacity!', 1
      ROLLBACK TRANSACTION
    END
END
```

6.2. Weryfikacja dostępnej liczby miejsc na warsztaty

```
CREATE TRIGGER [dbo].[check_if_workshop_reservation_exceeds_capacity]
  ON [dbo].[Workshop_Reservation]
  AFTER INSERT, UPDATE
AS
BEGIN
  SET NOCOUNT ON;
  DECLARE @workshop_ID int = (SELECT workshop_ID FROM inserted);
  DECLARE @reserved_tickets int = (SELECT sum(student_tickets +
  regular_tickets) FROM Workshop_Reservation WHERE workshop_ID = @workshop_ID
  and NOT EXISTS (SELECT * FROM Refund AS ref WHERE ref.reservation_ID =
  reservation_ID))
  IF (@reserved_tickets > (SELECT capacity FROM Workshop WHERE workshop_ID =
  @workshop_ID))
    BEGIN;
      THROW 51000, 'Requested number of tickets exceeds workshop''s
      capacity!',1
      ROLLBACK TRANSACTION
    END
END
```

6.3. Weryfikacja rezerwacji na warsztatach z rezerwacją na dany dzień

```
CREATE TRIGGER [dbo].[check_if_workshop_reservation_exceeds_day_reservation]
  ON [dbo].[Workshop_Reservation]
  AFTER INSERT, UPDATE
AS
BEGIN
  SET NOCOUNT ON;
  DECLARE @reservation_ID int = (SELECT reservation_ID FROM inserted);
  DECLARE @workshop_student_tickets int = (SELECT student_tickets FROM
  inserted);
  DECLARE @workshop_regular_tickets int = (SELECT regular_tickets FROM
  inserted);
  DECLARE @day_student_tickets int = (SELECT student_tickets FROM Reservation
  WHERE reservation_ID = @reservation_ID);
  DECLARE @day_regular_tickets int = (SELECT regular_tickets FROM Reservation
  WHERE reservation_ID = @reservation_ID);
  IF (@workshop_regular_tickets > @day_regular_tickets)
    BEGIN;
      THROW 51000, 'Requested number of regular workshop tickets exceeds the
      number of reserved regular day tickets!', 1
      ROLLBACK TRANSACTION
    END
  IF (@workshop_student_tickets > @day_student_tickets)
    BEGIN;
      THROW 51000, 'Requested number of student workshop tickets exceeds the
      number of reserved student day tickets!', 1
      ROLLBACK TRANSACTION
    END
END
```

6.4. Dodawanie dni konferencji z przeszłości lub za mniej niż tydzień

```
CREATE TRIGGER [check_if_conference_within_a_week]
  ON Conference_Day
  AFTER INSERT, UPDATE
AS
BEGIN
  SET NOCOUNT ON
  DECLARE @conference_date DATE = (SELECT date FROM inserted)
  IF (DATEDIFF(dd, GETDATE(), @conference_date) <= 7)
    BEGIN;
      THROW 51000, 'You cannot add a conference that''s less than a week
      ahead!', 1
      ROLLBACK TRANSACTION
    END
END
```

6.5. Dopuszczalność opłaty

```
CREATE TRIGGER [dbo].[check_if_payment_is_possible]
  ON [dbo].[Payment]
  AFTER INSERT, UPDATE
AS
BEGIN
  SET NOCOUNT ON;

  DECLARE @reservation_ID int = (SELECT reservation_ID FROM inserted);
  DECLARE @amount_paid money = (SELECT sum(amount_paid) FROM Payment WHERE
  reservation_ID = @reservation_ID);
  DECLARE @full_price money = (dbo.reservation_price (@reservation_ID));
  IF (@amount_paid > @full_price)
    BEGIN;
      THROW 51000, 'This payment exceeds the overall price of this
      reservation!', 1
      ROLLBACK TRANSACTION
    END
  IF EXISTS (SELECT * FROM Refund WHERE reservation_ID = @reservation_ID)
    BEGIN;
      THROW 51000, 'This reservation had been cancelled and a refund had been
      filed!', 1
      ROLLBACK TRANSACTION
    END
END
```

6.6. Dublowanie rezerwacji na dzień

```
CREATE TRIGGER [dbo].[check_if_client_already_has_day_reservation]
  ON [dbo].[Reservation]
  AFTER INSERT, UPDATE
AS
BEGIN
  SET NOCOUNT ON;
  DECLARE @reservation_ID int = (SELECT reservation_ID FROM inserted);
  DECLARE @conference_day_ID int = (SELECT conference_day_ID FROM inserted);
  DECLARE @client_ID int = (SELECT client_ID FROM inserted);
  IF EXISTS (SELECT * FROM Reservation WHERE reservation_ID <> @reservation_ID
  and conference_day_ID = @conference_day_ID and client_ID = @client_ID)
     BEGIN;
       THROW 51000, 'This client already has a reservation for this conference
       day!', 1
       ROLLBACK TRANSACTION
     END
END
```

6.7. Dublowanie rezerwacji na warsztatach

```
CREATE TRIGGER [dbo].[check_if_client_already_has_workshop_reservation]
  ON [dbo].[Workshop_Reservation]
  AFTER INSERT, UPDATE
AS
BEGIN
  SET NOCOUNT ON;
  DECLARE @reservation_ID int = (SELECT reservation_ID FROM inserted);
  DECLARE @workshop_reservation_ID int = (SELECT workshop_reservation_ID FROM
  inserted);
  DECLARE @workshop_ID int = (SELECT workshop_ID FROM inserted);
  DECLARE @client_ID int = (SELECT client_ID FROM Reservation WHERE
  reservation_ID = @reservation_ID);
  IF EXISTS (SELECT * FROM Workshop_Reservation WHERE workshop_reservation_ID
  <> @workshop_reservation_ID and workshop_ID = @workshop_ID and
  reservation_ID = @reservation_ID)
    BEGIN;
      THROW 51000, 'This client already has a reservation for this workshop!',
      1
      ROLLBACK TRANSACTION
    END
END
```

6.8. Przypisywanie uczestników do anulowanych rezerwacji

```
CREATE TRIGGER [dbo].[check_if_reservation_not_cancelled]
  ON [dbo].[Conference_Day_Attendee]
  AFTER INSERT, UPDATE
AS
BEGIN
 SET NOCOUNT ON;
 DECLARE @reservation_id int = (SELECT reservation_ID FROM inserted)
 IF EXISTS (SELECT * FROM Refund WHERE reservation_ID = @reservation_id)
  BEGIN;
    THROW 51000, 'This reservation had been cancelled!', 1
    ROLLBACK TRANSACTION
  END
END
```

6.9. Przypisywanie uczestników do nieopłaconych rezerwacji

```
CREATE TRIGGER [dbo].[check_if_reservation_paid]
  ON [dbo].[Conference_Day_Attendee]
  AFTER INSERT, UPDATE
AS
BEGIN
  SET NOCOUNT ON;
  DECLARE @reservation_id int = (SELECT reservation_ID FROM inserted)
  IF (dbo.reservation_price (@reservation_id) <> (SELECT sum(amount_paid) FROM
  Payment WHERE reservation_ID = @reservation_id))
    BEGIN;
      THROW 51000, 'This reservation has not been fully paid!', 1
      ROLLBACK TRANSACTION
    END
END
```

6.10. Dostępność progu cenowego w dniu rezerwacji

```
CREATE TRIGGER [dbo].[check_if_price_available]
  ON [dbo].[Reservation]
  AFTER INSERT, UPDATE
AS
BEGIN
  SET NOCOUNT ON;
  DECLARE @conference_day_id int = (SELECT conference_day_id FROM inserted);
  DECLARE @reservation_date date = (SELECT reservation_date FROM inserted);
  DECLARE @conference_id INT = (SELECT conference_id FROM Conference_Day WHERE
  conference_day_ID = @conference_day_id)
  IF NOT EXISTS (SELECT * FROM Price WHERE conference_ID = @conference_id and
  start_date <= @reservation_date and end_date >= @reservation_date)
    BEGIN;
      THROW 51000, 'No price point available for this conference', 1
      ROLLBACK TRANSACTION
    END
END
```

6.11. Weryfikacja przypisanych uczestników z biletami na dany dzień konferencji

```
CREATE TRIGGER [dbo].[check_if_day_attendee_acceptable]
  ON [dbo].[Conference_Day_Attendee]
  AFTER INSERT, UPDATE
AS
BEGIN
  SET NOCOUNT ON;
  DECLARE @reservation_id int = (SELECT reservation_ID FROM inserted)
  DECLARE @students_registered int = (SELECT count(*) FROM reservation AS r
  JOIN Conference_Day_Attendee AS cda ON cda.reservation_ID = r.reservation_ID
  JOIN Participant AS p ON p.participant_ID = cda.participant_ID WHERE
  p.student_ID IS NOT NULL and r.reservation_ID = @reservation_id);
  DECLARE @regular_registered int = (SELECT count(*) FROM reservation AS r
  JOIN Conference_Day_Attendee AS cda ON cda.reservation_ID = r.reservation_ID
  JOIN Participant AS p ON p.participant_ID = cda.participant_ID WHERE
  p.student_ID IS NULL and r.reservation_ID = @reservation_id);
  DECLARE @students_reserved int = (SELECT student_tickets FROM Reservation
  WHERE reservation_id = @reservation_id);
  DECLARE @regular_reserved int = (SELECT regular_tickets FROM Reservation
  WHERE reservation_id = @reservation_id)
  IF (@students_registered + @regular_registered > @students_reserved +
  @regular_reserved)
    BEGIN;
      THROW 51000, 'All participants have already been registered for this
      reservation!', 1
      ROLLBACK TRANSACTION
    END
  IF (@students_registered > @students_reserved)
    BEGIN;
      THROW 51000, 'All student participants have already been registered for
      this reservation!', 1
      ROLLBACK TRANSACTION
    END
  IF (@regular_registered > @regular_reserved)
    BEGIN;
      THROW 51000, 'All regular participants have already been registered for
      this reservation!', 1
      ROLLBACK TRANSACTION
    END
END
```

6.12. Weryfikacja liczby przypisanych uczestników z liczbą biletów na dany warsztat

```
CREATE TRIGGER [dbo].[check_if_workshop_attendee_acceptable]
  ON [dbo].[Workshop_Attendee]
  AFTER INSERT, UPDATE
AS
BEGIN
  SET NOCOUNT ON;

  DECLARE @workshop_reservation_id int = (SELECT workshop_reservation_ID FROM
  inserted);

  DECLARE @students_registered int = (SELECT count(*) FROM
  Workshop_Reservation AS wr JOIN Workshop_Attendee AS wa ON
  wa.workshop_reservation_ID = wr.workshop_reservation_ID JOIN
  Conference_Day_Attendee AS cda ON cda.conference_day_attendee_ID =
  wa.conference_day_attendee_id JOIN Participant AS p ON p.participant_ID =
  cda.participant_ID WHERE p.student_ID IS NOT NULL and
  wr.workshop_reservation_ID = @workshop_reservation_id);

  DECLARE @regular_registered int = (SELECT count(*) FROM Workshop_Reservation
  AS wr JOIN Workshop_Attendee AS wa ON wa.workshop_reservation_ID =
  wr.workshop_reservation_ID JOIN Conference_Day_Attendee AS cda ON
  cda.conference_day_attendee_ID = wa.conference_day_attendee_id JOIN
  Participant AS p ON p.participant_ID = cda.participant_ID WHERE p.student_ID
  IS NULL and wr.workshop_reservation_ID = @workshop_reservation_id);

  DECLARE @students_reserved int = (SELECT student_tickets FROM
  Workshop_Reservation WHERE workshop_reservation_ID =
  @workshop_reservation_id);

  DECLARE @regular_reserved int = (SELECT regular_tickets FROM
  Workshop_Reservation WHERE workshop_reservation_ID =
  @workshop_reservation_id)

  IF (@students_registered + @regular_registered > @students_reserved +
  @regular_reserved)
    BEGIN;
      THROW 51000, 'All participants have already been registered for this
      workshop reservation!', 1
      ROLLBACK TRANSACTION
    END
  IF (@students_registered > @students_reserved)
    BEGIN;
      THROW 51000, 'All student participants have already been registered for
      this workshop reservation!', 1
      ROLLBACK TRANSACTION
    END
  IF (@regular_registered > @regular_reserved)
```

```
    BEGIN;
      THROW 51000, 'All regular participants have already been registered for
      this workshop reservation!', 1
      ROLLBACK TRANSACTION
    END
END
```

6.13. Kolizja warsztatów dla danego uczestnika

```
CREATE TRIGGER [dbo].[check_for_workshop_overlap]
  ON [dbo].[Workshop_Attendee]
  AFTER INSERT, UPDATE
AS
BEGIN
  SET NOCOUNT ON;
  DECLARE @workshop_reservation_id int = (SELECT workshop_reservation_ID FROM
  inserted);
  DECLARE @reservation_id int = (SELECT reservation_ID FROM
  Workshop_Reservation WHERE workshop_reservation_ID =
  @workshop_reservation_id);
  DECLARE @conference_day_attendee_ID int = (SELECT conference_day_attendee_id
  FROM inserted);
  DECLARE @workshop_id int = (SELECT workshop_id FROM Workshop_Reservation
  WHERE workshop_reservation_ID = @workshop_reservation_id);
  DECLARE @start_time time(0) = (SELECT start_time FROM Workshop WHERE
  workshop_ID = @workshop_id);
  DECLARE @end_time time(0) = (SELECT end_time FROM Workshop WHERE workshop_ID
  = @workshop_id);

  IF EXISTS (SELECT * FROM Workshop_Attendee WHERE conference_day_attendee_id
  = @conference_day_attendee_ID and workshop_reservation_ID in (SELECT
  workshop_reservation_ID FROM Workshop_Reservation WHERE reservation_ID =
  @reservation_id and workshop_reservation_ID <> @workshop_reservation_id
  and workshop_id in ((SELECT workshop_id FROM Workshop WHERE start_time <=
  @end_time and end_time >= @start_time) union all (SELECT workshop_id FROM
  Workshop WHERE start_time <= @end_time and @start_time <= end_time))))

    BEGIN;
      THROW 51000, 'This participant had already been registered for a
      workshop at the same time', 1
      ROLLBACK TRANSACTION
    END
END
```

6.14. Kolizja progów cenowych

```
CREATE TRIGGER [dbo].[check_for_price_overlap]
  ON [dbo].[Price]
  AFTER INSERT, UPDATE
AS
BEGIN
  SET NOCOUNT ON;
  DECLARE @conference_id int = (SELECT conference_id FROM inserted);
  DECLARE @price_ID int = (SELECT price_id FROM inserted);
  DECLARE @start_date date = (SELECT start_date FROM inserted);
  DECLARE @end_date date = (SELECT end_date FROM inserted);
  IF EXISTS (SELECT * FROM Price WHERE conference_ID = @conference_id and
  price_id <> @price_ID and price_id in ((SELECT price_id FROM Price WHERE
  start_date <= @start_date and end_date >= @start_date) UNION ALL (SELECT
  price_id FROM Price WHERE end_date <= @end_date and end_date >= @end_date)
  UNION ALL (SELECT price_id FROM Price WHERE start_date >= @start_date and
  end_date <= @end_date)))
    BEGIN;
      THROW 51000, 'This price overlaps with another price added to this
      conference', 1
      ROLLBACK TRANSACTION
    END
END
```

6.15. Zgodność dat progów cenowych z datą konferencji

```
CREATE TRIGGER [dbo].[check_if_price_before_conference]
  ON [dbo].[Price]
  AFTER INSERT, UPDATE
AS
BEGIN
   SET NOCOUNT ON
   DECLARE @conference_id INT = (SELECT conference_ID FROM inserted);
   DECLARE @end_date DATE = (SELECT Inserted.end_date FROM Inserted);
   DECLARE @start_of_conference DATE = (SELECT min(date) FROM Conference_Day
   WHERE conference_ID = @conference_id)
   IF (DATEDIFF(dd, @end_date, @start_of_conference) < 7)
     BEGIN;
       THROW 51000, 'You cannot add a price after the conference or throughout
       the last week before the conference!', 1
       ROLLBACK TRANSACTION
     END
END
```

7. Funkcje

7.1. Łączna cena danej rezerwacji

```
CREATE FUNCTION [dbo].[reservation_price](
 @reservation_id int
) RETURNS MONEY
AS
BEGIN
 IF NOT EXISTS (SELECT * FROM Reservation WHERE reservation_ID =
 @reservation_id)
   BEGIN
     RETURN 0
   END
 ELSE
   BEGIN
     DECLARE @conference_day_id INT = (SELECT conference_day_id FROM
     dbo.Reservation WHERE reservation_ID = @reservation_id);
     DECLARE @conference_id INT = (SELECT conference_id FROM
     dbo.Conference_Day WHERE conference_day_ID = @conference_day_id);
     DECLARE @student_disount FLOAT = (SELECT student_discount FROM
     dbo.Conference WHERE conference_ID = @conference_id);
     DECLARE @day_student_tickets INT = (SELECT student_tickets FROM
     dbo.Reservation WHERE reservation_ID = @reservation_id);
     DECLARE @day_regular_tickets INT = (SELECT regular_tickets FROM
     dbo.Reservation WHERE reservation_ID = @reservation_id);
     DECLARE @reservation_date DATE = (SELECT reservation_date FROM
     dbo.Reservation WHERE reservation_ID = @reservation_id);
     DECLARE @day_price MONEY = (SELECT price FROM Price WHERE start_date <=
     @reservation_date and end_date >= @reservation_date);
     DECLARE @day_price_sum money = COALESCE(((((1 - @student_disount) *
     @day_student_tickets) + @day_regular_tickets) * @day_price), 0);
     DECLARE @workshops_price_sum MONEY = COALESCE(((((SELECT
     sum(wr.student_tickets * w.price) FROM Workshop_Reservation AS wr JOIN
     Workshop AS w ON w.workshop_ID = wr.workshop_ID WHERE wr.reservation_ID
     = @reservation_ID) * (1 - @student_disount))) + (SELECT
     sum(wr.regular_tickets * w.price) FROM Workshop_Reservation AS wr JOIN
     Workshop AS w ON w.workshop_ID = wr.workshop_ID WHERE reservation_ID =
     @reservation_ID)), 0);
   END
   RETURN (@day_price_sum + @workshops_price_sum)
END
```

7.2. Liczba wolnych miejsc na danym warsztacie

```
ALTER FUNCTION [dbo].[available_workshop_tickets](
  @workshopID int
) RETURNS INT
AS
BEGIN
  IF NOT EXISTS (SELECT * FROM Workshop WHERE workshop_ID = @workshopID)
    BEGIN
      RETURN 0
    END
  DECLARE @return INT = ((select w.capacity from Workshop as w where
  w.workshop_ID = @workshopID)
  - COALESCE((select sum(wr.student_tickets + wr.regular_tickets) from
  Workshop_Reservation as wr
  where wr.workshop_ID = @workshopID),0)
  + COALESCE((select sum(wr.student_tickets + wr.regular_tickets) from
  Workshop_Reservation as wr
  where wr.workshop_ID = @workshopID and wr.reservation_ID in (select
  reservation_ID from Refund)),0))
  RETURN @return
END
```

7.3. Liczba wolnych miejsc w danym dniu konferencji

```
CREATE FUNCTION [dbo].[available_conference_day_tickets](
  @conference_day_ID int
) RETURNS INT
AS
BEGIN
  IF NOT EXISTS (SELECT * FROM Conference_Day WHERE conference_day_ID =
  @conference_day_ID)
    BEGIN
    RETURN 0
  END
  DECLARE @return INT = ((SELECT capacity FROM Conference_Day WHERE
  conference_day_ID = @conference_day_ID)
  - COALESCE((SELECT sum(student_tickets + regular_tickets) FROM Reservation
  WHERE conference_day_ID = @conference_day_ID), 0)
  + COALESCE((SELECT sum(student_tickets + regular_tickets) FROM Reservation
  WHERE conference_day_ID = @conference_day_ID
  and reservation_ID in (SELECT reservation_ID FROM Refund)), 0))
  RETURN @return
END
```

8. Widoki

8.1. Wszystkie konferencje wraz z datami

```
CREATE VIEW Conferences_Summary AS
SELECT
c.conference_name AS [Conference name],
min(date) AS [Start date],
max(date) AS [End date],
cast(c.conference_description AS nvarchar(max)) AS Description
FROM Conference AS c
JOIN Conference_Day AS cd
on cd.conference_ID = c.conference_ID
GROUP BY c.conference_ID, c.conference_name, cast(c.conference_description AS
nvarchar(max))
```

8.2. Przyszłe konferencje

```
CREATE VIEW Future_Conferences AS
SELECT * FROM Conferences_Summary
WHERE [Start date] > GETDATE()
```

8.3. Szczegóły poszczególnych rezerwacji

```
CREATE VIEW Reservations_Summary AS
SELECT c.name as [Client],
cnf.conference_name + ' | ' + convert(nvarchar(10), cd.date) as [Conference],
r.reservation_date as [Reservation date],
dbo.reservation_price(r.reservation_id) AS [Full price],
(COALESCE((SELECT sum(amount_paid) FROM Payment WHERE reservation_ID =
r.reservation_ID),0)) AS [Amount paid],
(SELECT CASE WHEN COUNT(*) > 0 THEN ref.amount_refunded ELSE '-' END AS Expr1
FROM Refund AS ref WHERE (ref.reservation_ID = r.reservation_ID) GROUP BY
amount_refunded) AS [Amount refunded]
FROM Reservation AS r
JOIN Client AS c
ON c.client_ID = r.client_ID
JOIN Conference_Day as cd
on cd.conference_day_ID = r.conference_day_ID
JOIN Conference as cnf
on cnf.conference_ID = r.conference_day_ID
```

8.4. Uczestnicy dni konferencji

```
CREATE VIEW Conference_Day_Attendee_List AS
SELECT c.conference_name AS [Conference], cd.date AS [Date],
p.first_name AS [First name],
 p.last_name AS [Last name]
FROM Participant AS p
JOIN dbo.Conference_Day_Attendee AS cda
ON p.participant_ID = cda.participant_ID
JOIN Reservation AS r
ON cda.reservation_ID = r.reservation_ID
JOIN Conference_Day AS cd
ON r.conference_day_ID = cd.conference_day_ID
JOIN Conference AS c
on c.conference_ID = cd.conference_ID
GROUP BY c.conference_ID, c.conference_name, cd.conference_day_ID, cd.date,
p.first_name, p.last_name
```

8.5. Uczestnicy warsztatów

```
CREATE VIEW Workshop_Attendee_List AS
SELECT c.conference_name AS [Conference name], cd.date AS [Date],
w.workshop_name AS [Workshop name], w.start_time AS [Start time],
p.first_name AS [First name], p.last_name AS [Last name]
FROM Participant AS p
JOIN Conference_Day_Attendee AS cda
ON p.participant_ID = cda.participant_ID
JOIN Workshop_Attendee AS wa
ON wa.conference_day_attendee_id = cda.conference_day_attendee_ID
JOIN Workshop_Reservation AS wr
ON cda.reservation_ID = wr.reservation_ID and wa.workshop_reservation_ID =
wr.workshop_reservation_ID
JOIN Workshop AS w
on w.workshop_id = wr.workshop_ID
JOIN Conference_Day AS cd
on cd.conference_day_ID = w.conference_day_ID
JOIN Conference AS c
on c.conference_ID = cd.conference_ID
```

8.6. Przyszłe warsztaty oraz dostępna liczba biletów

```
CREATE VIEW Available_Workshops AS
SELECT c.conference_name AS [Conference], cd.date AS [Date], w.workshop_name
AS [Workshop name],
(convert(varchar(10),
(w.capacity - sum (wr.student_tickets+wr.regular_tickets))) + '/' + convert
(varchar(10), w.capacity))
AS [Available tickets], cast(w.description AS nvarchar(max)) AS [Workshop
description]
FROM dbo.Workshop AS w
JOIN dbo.Workshop_Reservation AS wr
ON w.workshop_ID = wr.workshop_ID
JOIN Conference_Day AS cd
on cd.conference_day_ID = w.conference_day_ID
JOIN Conference AS c
on c.conference_ID = cd.conference_ID
WHERE wr.reservation_ID not in (SELECT reservation_ID FROM Refund) and
cd.date >= GETDATE()
GROUP BY w.workshop_ID, w.workshop_name, w.capacity, c.conference_ID,
c.conference_name, cast(w.description AS nvarchar(max)), cd.date
```

8.7. Łączne wpłaty poszczególnych klientów

```
CREATE VIEW [Client_Payments] AS
SELECT c.name as [Name],
c.phone_number as [Phone number],
((SELECT sum(amount_paid) FROM Payment WHERE reservation_ID in
(SELECT reservation_ID FROM Reservation WHERE client_ID = c.client_ID))
- COALESCE((SELECT amount_refunded FROM Refund WHERE reservation_ID in
(SELECT reservation_ID FROM Reservation WHERE client_ID = c.client_ID)),0))
as [Sum of all payments]
FROM Client as c
```

8.8. Najpopularniejsze warsztaty

```
CREATE VIEW [Workshop_Popularity] AS
SELECT w.workshop_name as [Name],
COALESCE ((SELECT sum(student_tickets + regular_tickets)
FROM Workshop_Reservation as wr WHERE w.workshop_ID = wr.workshop_ID
and not exists (SELECT * FROM refund WHERE reservation_ID =
wr.reservation_ID)), 0) as [All tickets]
FROM Workshop as w
```

9. Widoki parametryzowane

9.1. Lista klientów, którzy nie uzupełnili w pełni swoich list uczestników w danym dniu konferencji

```
CREATE PROCEDURE [dbo].[conference_day_missing_attendees](
  @conference_id int, @date date
) AS BEGIN
  IF @conference_id IS NULL or @date IS NULL
    BEGIN
      RAISERROR ('Null values are not allowed', 14,1)
      RETURN
    END
  IF NOT EXISTS (SELECT * FROM Conference WHERE conference_ID =
  @conference_id)
    BEGIN
      RAISERROR ('This conference doesn''t exist!', 14,1)
      RETURN
    END
  IF NOT EXISTS (SELECT * FROM Conference_Day WHERE conference_ID =
  @conference_id and date = @date)
    BEGIN
      RAISERROR ('This conference doesn''t include this date!', 14,1)
      RETURN
    END
  DECLARE @conference_day_id int = (SELECT conference_day_id FROM
  Conference_Day WHERE conference_ID = @conference_id and date = @date);
  SELECT (SELECT name FROM Client AS c WHERE c.client_ID = r.client_ID) AS
  [Client name],
  (SELECT c.phone_number FROM Client AS c WHERE c.client_ID = r.client_ID) AS
  [Phone number],
  CONVERT(varchar(10), (SELECT count(*) FROM Conference_Day_Attendee AS cda
  JOIN participant AS p ON p.participant_ID = cda.participant_ID WHERE
  cda.reservation_ID = r.reservation_ID and p.student_ID IS NOT NULL)) + '/' +
  CONVERT(varchar(10), student_tickets) AS [Registered student participants],
  CONVERT(varchar(10), (SELECT count(*) FROM Conference_Day_Attendee AS cda
  JOIN participant AS p ON p.participant_ID = cda.participant_ID WHERE
  cda.reservation_ID = r.reservation_ID and p.student_ID IS NULL)) + '/' +
  CONVERT(varchar(10), regular_tickets) AS [Registered regular participants]
  FROM Reservation AS r
  GROUP BY r.client_id, r.reservation_ID, r.regular_tickets, r.student_tickets
  HAVING (regular_tickets) <> (SELECT count(*) FROM Conference_Day_Attendee AS
  cda JOIN participant AS p ON p.participant_ID = cda.participant_ID WHERE
  cda.reservation_ID = r.reservation_ID and p.student_ID IS NULL)
  or (student_tickets) <> (SELECT count(*) FROM Conference_Day_Attendee AS cda
  JOIN participant AS p ON p.participant_ID = cda.participant_ID WHERE
  cda.reservation_ID = r.reservation_ID and p.student_ID IS NOT NULL)
END
```

9.2. Lista warsztatów z nieuzupełnionymi miejscami dla danej rezerwacji

```
CREATE PROCEDURE [dbo].[workshop_missing_attendees](
  @reservation_id int
) AS
BEGIN
  IF @reservation_id IS NULL
    BEGIN
      RAISERROR ('Null values are not allowed', 14,1)
      RETURN
    END
  IF NOT EXISTS (SELECT * FROM Reservation WHERE reservation_ID =
  @reservation_id)
    BEGIN
      RAISERROR ('This reservation doesn''t exist!', 14,1)
      RETURN
    END
  SELECT w.workshop_name,
  CONVERT(varchar(10), (SELECT count(*) FROM Workshop_Attendee AS wa
  JOIN Conference_Day_Attendee AS cda ON cda.conference_day_attendee_ID =
  wa.conference_day_attendee_id JOIN participant AS p ON p.participant_ID =
  cda.participant_ID WHERE wa.workshop_reservation_ID =
  wr.workshop_reservation_ID and p.student_ID IS NOT NULL)) + '/' +
  CONVERT(varchar(10), student_tickets) AS [Registered student participants],
  CONVERT(varchar(10), (SELECT count(*) FROM Workshop_Attendee AS wa JOIN
  Conference_Day_Attendee AS cda ON cda.conference_day_attendee_ID =
  wa.conference_day_attendee_id JOIN participant AS p ON p.participant_ID =
  cda.participant_ID WHERE wa.workshop_reservation_ID =
  wr.workshop_reservation_ID and p.student_ID IS NULL)) + '/' +
  CONVERT(varchar(10), regular_tickets) AS [Registered regular participants]
  FROM Workshop_Reservation AS wr
  JOIN workshop AS w
  on w.workshop_id = wr.workshop_id
  WHERE wr.reservation_ID = @reservation_id
  GROUP BY w.workshop_ID, w.workshop_name, wr.student_tickets,
  wr.regular_tickets , wr.workshop_reservation_ID
  HAVING (regular_tickets) <> (SELECT count(*) FROM Workshop_Attendee AS wa
  JOIN Conference_Day_Attendee AS cda ON cda.conference_day_attendee_ID =
  wa.conference_day_attendee_id JOIN participant AS p ON p.participant_ID =
  cda.participant_ID WHERE wa.workshop_reservation_ID =
  wr.workshop_reservation_ID and p.student_ID IS NULL)
  or (student_tickets) <> (SELECT count(*) FROM Workshop_Attendee AS wa
  JOIN Conference_Day_Attendee AS cda ON cda.conference_day_attendee_ID =
  wa.conference_day_attendee_id JOIN participant AS p ON p.participant_ID =
  cda.participant_ID WHERE wa.workshop_reservation_ID =
  wr.workshop_reservation_ID and p.student_ID IS NOT NULL)
END
```

9.3. Lista uczestników konferencji

```
CREATE PROCEDURE [dbo].[conference_attendees_print_ids](
  @conference_id int
) AS
BEGIN
  IF @conference_id IS NULL
    BEGIN
      RAISERROR ('Null values are not allowed', 14,1)
      RETURN
    END
  IF NOT EXISTS (SELECT * FROM Conference WHERE conference_id =
  @conference_id)
    BEGIN
      RAISERROR ('This conference doesn''t exist!', 14,1)
      RETURN
    END
  SELECT p.first_name as [First name], p.last_name as [Last name],
  (SELECT CASE WHEN tax_id_number IS NOT NULL THEN name ELSE NULL END
  FROM client as c JOIN Reservation as r ON r.client_ID = c.client_ID
  JOIN Conference_Day_Attendee as cda ON cda.reservation_ID = r.reservation_ID
  WHERE cda.participant_ID = p.participant_id) as [Company]
  FROM Participant as p
  WHERE p.participant_id in (SELECT participant_id FROM
  Conference_Day_Attendee WHERE reservation_ID in (SELECT reservation_ID FROM
  Reservation WHERE conference_day_ID in (SELECT conference_day_ID FROM
  Conference_Day WHERE conference_ID = @conference_id)))
END
```

10. Indeksy

Indeksy zostały założone na kluczach obcych występujących w tabelach.

```
CREATE INDEX conference_day_conference ON Conference_Day (conference_ID)
CREATE INDEX conference_day_attendee_reservation ON dbo.Conference_Day_Attendee
(reservation_ID)
CREATE INDEX conference_day_attendee_participant ON dbo.Conference_Day_Attendee
(participant_ID)
CREATE INDEX payment_reservation ON dbo.Payment (reservation_ID)
CREATE INDEX price_conference ON dbo.Price (conference_ID)
CREATE INDEX refund_reservation ON dbo.Refund (reservation_ID)
CREATE INDEX reservation_client ON dbo.Client (client_ID)
CREATE INDEX reservation_conference_day ON dbo.Reservation (conference_day_ID)
CREATE INDEX workshop_conference_day ON dbo.Workshop (conference_day_ID)
CREATE INDEX workshop_attendee_conference_attendee ON dbo.Workshop_Attendee
(conference_day_attendee_id)
CREATE INDEX workshop_attendee_workshop_reservation ON dbo.Workshop_Attendee
(workshop_reservation_ID)
CREATE INDEX workshop_reservation_reservation ON dbo.Workshop_Reservation
(reservation_ID)
CREATE INDEX workshop_reservation_workshop ON dbo.Workshop_Reservation (workshop_ID)
```

11. Generator

Do wygenerowania skryptów zapełniających bazę wykorzystany został program napisany w Javie oraz procedury dostępne w bazie danych. Na potrzeby generowania danych utworzone zostały osobne procedury pozwalające na dodanie, opłacanie i anulowanie rezerwacji z przeszłości oraz dodanie progów cenowych do konferencji z przeszłości. Aby skrypt dodający do bazy wygenerowane dane działał poprawnie, konieczne jest usunięcie danych występujących w bazie oraz wyzerowanie kluczy głównych.

Wygenerowane dane odpowiadają trzem latom działalności firmy - każda konferencja ma 2-3 dni, a w każdym dniu organizowane są 4 warsztaty. Liczba klientów i zarejestrowanych przez nich uczestników jest randomizowana, jednak spełnia zadany warunek średnio dwustu uczestników na konferencję.

12. Uprawnienia

12.1. Klienci

- ❏ rezerwowanie miejsc na konferencji
- ❏ rezerwowanie miejsc na warsztatach
- ❏ rejestrowanie uczestników na konferencji
- ❏ rejestrowanie uczestników na warsztatach
- ❏ dodawanie biletów do rezerwacji
- ❏ przeglądanie przyszłych konferencji
- ❏ przeglądanie historii rezerwacji

12.2. Uczestnicy

- ❏ dodawanie danych osobowych do rezerwacji na dzień konferencji
- ❏ dodawanie danych osobowych do rezerwacji na warsztaty
- ❏ podgląd swoich rejestracji

12.3. Pracownicy firmy

- ❏ dodawanie konferencji
- ❏ dodawanie dni konferencji
- ❏ dodawanie warsztatów
- ❏ dodawanie progów cenowych
- ❏ anulowanie rezerwacji
- ❏ dostęp do widoków informacyjnych

12.4. Administrator bazy - pełny dostęp do bazy