

Projekt:

Konferencje

Oscar Teeninga & Kamil Szarek

Podstawy baz danych

Celem projektu jest zaprojektowanie i zaimplementowanie systemu bazodanowego dla firmy organizującej konferencje.

**Spis treści**

[Schemat bazy danych 5](file:///C:\Users\kamsz\Desktop\BD\Projekt.docx#_Toc535959862)

[Tabele 6](#_Toc535959863)

[client 6](#_Toc535959864)

[reservation 6](#_Toc535959865)

[payment 7](#_Toc535959866)

[attendee 7](#_Toc535959867)

[conference\_attendee\_list 7](#_Toc535959868)

[workshop\_attendee\_list 8](#_Toc535959869)

[conference 8](#_Toc535959870)

[workshop\_reservation 8](#_Toc535959871)

[conference\_day 9](#_Toc535959872)

[workshop 9](#_Toc535959873)

[price 10](#_Toc535959874)

[Zainicjowanie bazy 11](#_Toc535959875)

[Usunięcie tabel o korelujących nazwach 11](#_Toc535959876)

[Wprowadzenie kluczy obcych 12](#_Toc535959877)

[Funkcje 13](#_Toc535959878)

[conference\_days 13](#_Toc535959879)

[conference\_days\_cost 13](#_Toc535959880)

[workshop\_cost 13](#_Toc535959881)

[places\_reserved\_on\_conference 14](#_Toc535959882)

[places\_available\_on\_conference 14](#_Toc535959883)

[places\_reserved\_on\_workshop 14](#_Toc535959884)

[places\_available\_on\_workshop 15](#_Toc535959885)

[conference\_attendees 15](#_Toc535959886)

[conference\_day\_attendees 16](#_Toc535959887)

[workshop\_attendees 16](#_Toc535959888)

[client\_payments 17](#_Toc535959889)

[numer\_of\_students 17](#_Toc535959890)

[Procedury 18](#_Toc535959891)

[add\_conference 18](#_Toc535959892)

[add\_client 18](#_Toc535959893)

[add\_payment 19](#_Toc535959894)

[add\_conference\_day 19](#_Toc535959895)

[add\_workshop 20](#_Toc535959896)

[add\_price 20](#_Toc535959897)

[add\_price\_in\_range 21](#_Toc535959898)

[add\_attendee 22](#_Toc535959899)

[add\_attendee\_on\_conference 22](#_Toc535959900)

[add\_attendee\_on\_workshop 23](#_Toc535959901)

[new\_conference\_reservation 24](#_Toc535959902)

[new\_workshop\_reservation 25](#_Toc535959903)

[update\_conference\_participants\_limit 25](#_Toc535959904)

[update\_workshop\_participants\_limit 26](#_Toc535959905)

[cancel\_reservation 26](#_Toc535959906)

[cancel\_workshop 27](#_Toc535959907)

[cancel\_all\_unpaid\_reservations 28](#_Toc535959908)

[cancel\_unpaid\_reservations\_in\_time 28](#_Toc535959909)

[Widoki 29](#_Toc535959910)

[most\_active\_clients 29](#_Toc535959911)

[available\_conferences 29](#_Toc535959912)

[available\_workshops 29](#_Toc535959913)

[clients\_with\_payment\_deficit 30](#_Toc535959914)

[cancelled\_reservations 30](#_Toc535959915)

[cancelled\_workshop\_reservations 30](#_Toc535959916)

[clients\_to\_call 30](#_Toc535959917)

[monthly\_income 31](#_Toc535959918)

[company\_clients 31](#_Toc535959919)

[attendee\_stats 31](#_Toc535959920)

[unpaid\_reservations 31](#_Toc535959921)

[clients\_with\_less\_attendees\_than\_reserved 32](#_Toc535959922)

[clients\_with\_less\_attendees\_than\_reserved\_for\_workshop 32](#_Toc535959923)

[Triggery 33](#_Toc535959924)

[checking\_number\_of\_reserved\_places 33](#_Toc535959925)

[checking\_number\_of\_reserved\_places\_workshop 33](#_Toc535959926)

[too\_many\_attendees\_for\_conference 33](#_Toc535959927)

[too\_many\_attendees\_for\_workshop 34](#_Toc535959928)

[max\_participants\_for\_workshop 34](#_Toc535959929)

[checking\_number\_of\_reserved\_places\_for\_individual\_client 34](#_Toc535959930)

[checking\_date 35](#_Toc535959931)

[checking\_date\_for\_workshop 35](#_Toc535959932)

[reservation\_cancel 35](#_Toc535959933)

[Indeksy 36](#_Toc535959934)

[Role w systemie 37](#_Toc535959935)

[Administrator systemu 37](#_Toc535959936)

[Pracownik firmy 37](#_Toc535959937)

[Klient 37](#_Toc535959938)

[Uczestnik 37](#_Toc535959939)

[Analiza wymagań 38](#_Toc535959940)

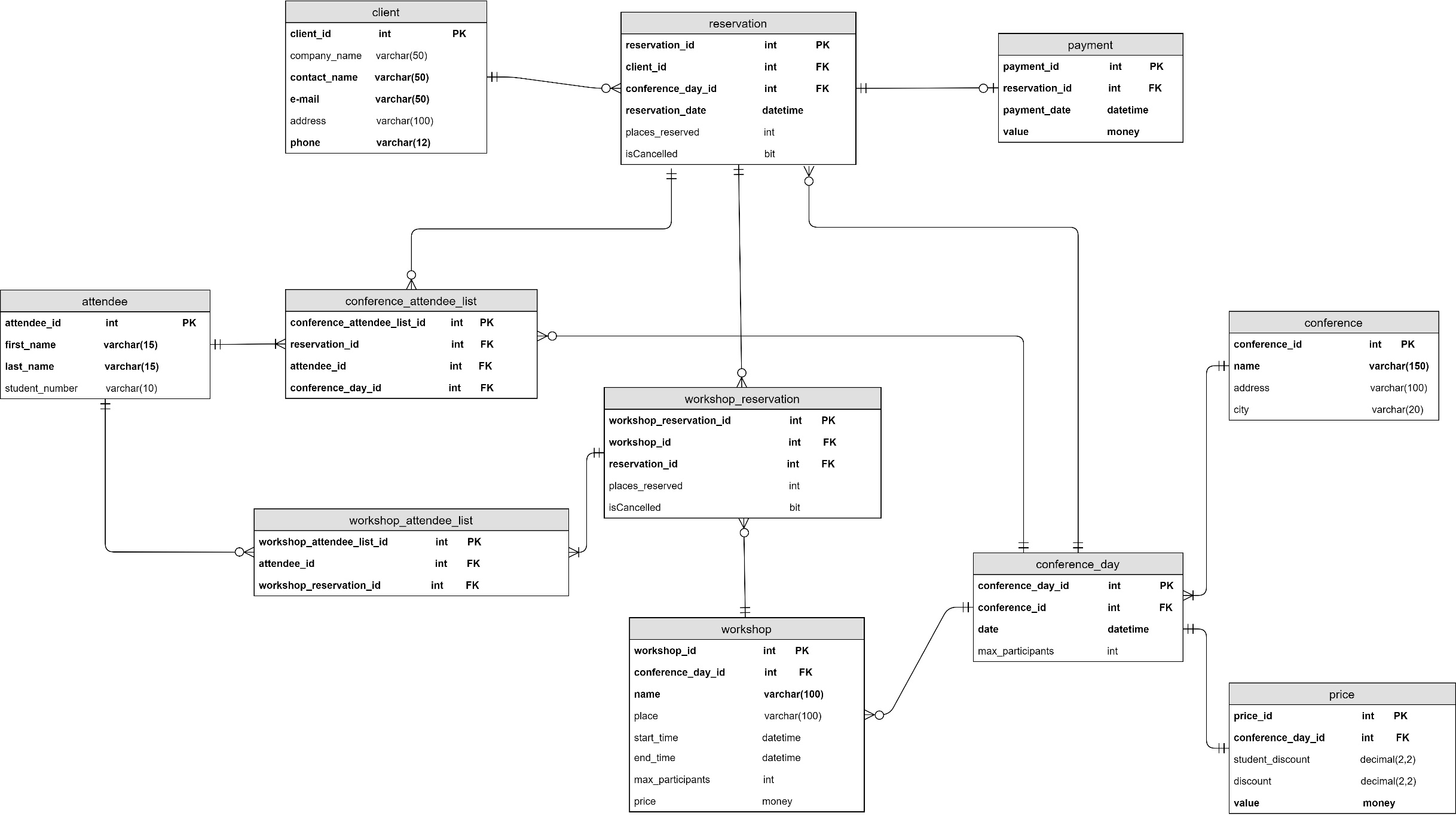
[Klient 38](#_Toc535959941)

[Klient (firma) 38](#_Toc535959942)

[Uczestnik 38](#_Toc535959943)

[Organizator 38](#_Toc535959944)

[Generator 39](#_Toc535959945)



# Schemat bazy danych

# Tabele

## client

Tabela przechowuje informacje o klientach korzystających z usług systemu. Zawiera niezbędne dane do skontaktowania się z klientem. Opcjonalne pole company\_name pozwala skategoryzować klientów na indywidualnych i firmowych.

*client\_id* identyfikator klienta (klucz główny),

*company\_name* nazwa firmy, będącej klientem,

*contact\_name* nazwisko przedstawiciela firmy / klienta indywidualnego,

*e-mail*  adres mailowy klienta,

*address*  adres klienta,

*phone*  telefon kontaktowy klienta

CREATE TABLE client (

  client\_id integer NOT NULL CONSTRAINT client\_pk PRIMARY KEY identity,

  company\_name varchar(50),

  contact\_name varchar(50) NOT NULL,

  [e-mail] varchar(50) NOT NULL CHECK ( [e-mail] IS NULL OR [e-mail] LIKE '%\_@%\_.\_\_%' ),,

  address varchar(100),

  phone varchar(12) NOT NULL

);

## reservation

Tabela przechowuje informacje o rezerwacjach klientów na dany dzień konferencji.

*reservation\_id* identyfikator rezerwacji (klucz główny),

*client\_id* identyfikator klienta (klucz obcy),

*conference\_day\_id* identyfikator dnia konferencji, którego dotyczy rezerwacja (klucz obcy)

*reservation\_date* data dokonania rezerwacji,

*places\_reserved* ilość zarezerwowanych miejsc,

*isCancelled* oznaczenie, czy dana rezerwacja została odwołana,

CREATE TABLE reservation (

  reservation\_id integer NOT NULL CONSTRAINT reservation\_pk PRIMARY KEY identity,

  client\_id integer NOT NULL,

  conference\_day\_id integer NOT NULL,

  reservation\_date datetime NOT NULL,

  places\_reserved integer,

  isCancelled bit NOT NULL default 0,

);

## payment

Tabela przechowuje informacje o płatnościach uiszczonych w ramach rezerwacji.

*payment\_id* identyfikator płatności (klucz główny),

*reservation\_id* identyfikator rezerwacji, za którą zapłacono (klucz obcy),

*payment\_date* data dokonania płatności,

*value*  wpłacona kwota,

CREATE TABLE payment (

  payment\_id integer NOT NULL CONSTRAINT payment\_pk PRIMARY KEY identity,

  reservation\_id integer NOT NULL,

  payment\_date datetime NOT NULL,

  value money NOT NULL

);

## attendee

Tabela przechowująca informacje o uczestnikach warsztatów i konferencji. Pole student\_number pozwala opcjonalnie wprowadzić numer indeksu studenta, co jednocześnie jest informacją, że mamy do czynienia z studentem.

*attendee\_id* identyfikator uczestnika (klucz główny),

*first\_name*  imię uczestnika,

*last\_name*  nazwisko uczestnika,

*student\_number* numer legitymacji studenckiej,

CREATE TABLE attendee (

  attendee\_id integer NOT NULL CONSTRAINT attendee\_pk PRIMARY KEY identity,

  first\_name varchar(15) NOT NULL,

  last\_name varchar(15) NOT NULL,

  student\_number varchar(10)

);

## conference\_attendee\_list

Tabela przechowuje uczestników danego dnia konferencji. Jest to tabela łącznikowa.

*conference\_attendee\_list\_id* identyfikator uczestnika konferencji (klucz główny),

*reservation\_id*  identyfikator rezerwacji, z którą dany uczestnik jest powiązany (obcy)

*attendee\_id* identyfikator uczestnika (klucz obcy),

*conference\_day\_id* identyfikator dnia konferencji, w którym uczestnik może wziąć

udział (klucz obcy),

CREATE TABLE conference\_attendee\_list (

  conference\_attendee\_list\_id integer NOT NULL CONSTRAINT conderence\_attendee\_list\_pk PRIMARY KEY identity,

  reservation\_id integer NOT NULL,

  attendee\_id integer NOT NULL,

  conference\_day\_id integer NOT NULL

);

## workshop\_attendee\_list

Tabela przechowuje uczestników danego warsztatu. Jest to tabela łącznikowa.

*workshop\_attendee\_list\_id* unikalny identyfikator uczestnika warsztatów (klucz główny),

*attendee\_id* identyfikator uczestnika (klucz obcy),

*workshop\_id* identyfikator warsztatu (klucz obcy),

CREATE TABLE workshop\_attendee\_list (

  workshop\_attendee\_list\_id integer NOT NULL CONSTRAINT workshop\_attendee\_list\_pk PRIMARY KEY identity,

  attendee\_id integer NOT NULL,

  workshop\_reservation\_id integer NOT NULL

);

## conference

Tabela przechowuje informacje o konkretnej konferencji.

*conference\_id*  identyfikator konferencji (klucz główny),

*name*  nazwa konferencji,

*address*  adres, pod którym dana konferencja się odbędzie,

*city*  miasto, w którym dana konferencja jest organizowana,

CREATE TABLE conference (

  conference\_id integer NOT NULL CONSTRAINT conference\_pk PRIMARY KEY identity,

  name varchar(150) NOT NULL,

  address varchar(100) NOT NULL,

  city varchar(20) NOT NULL

);

## workshop\_reservation

Tabela przechowuje informacje o zarezerwowanych warsztatach w ramach danej rezerwacji. Pole isCancelled pozwala anulować warsztat bez usuwania danej pozycji z tabeli.

*workshop\_reservation\_id* identyfikator rezerwacji (klucz główny),

*workshop\_id* identyfikator warsztatu (klucz obcy),

*places\_reserved* ilość zarezerwowanych miejsc,

*isCancelled* oznaczenie, czy dana rezerwacja została odwołana,

CREATE TABLE workshop\_reservation (

  workshop\_reservation\_id integer NOT NULL CONSTRAINT workshop\_reservation\_pk PRIMARY KEY identity,

  workshop\_id integer NOT NULL,

  reservation\_id integer NOT NULL,

  places\_reserved integer,

  isCancelled bit default 0

);

## conference\_day

Tabela przechowuje informacje o danym dniu konferencji na który mogą zapisywać się uczestnicy. W danym dniu konferencji może uczestniczyć ograniczona liczba osób określana przez pole max\_participants.

*conference\_day\_id* identyfikator danego dnia konferencji (klucz główny),

*conference\_id* identyfikator konferencji (klucz obcy),

*date*  dzień i godzina rozpoczęcia konferencji w danym dniu,

*max\_participants* maksymalna ilość uczestników konferencji,

CREATE TABLE conference\_day (

  conference\_day\_id integer NOT NULL CONSTRAINT conference\_day\_pk PRIMARY KEY identity,

  conference\_id integer NOT NULL,

  [date] datetime NOT NULL,

  max\_participants integer DEFAULT 1000

);

## workshop

Tabela przechowuje informacje o cenie uczestnictwa danego dnia konferencji. Przewidziane są pola pozwalające uwzględnić rabat oraz uglę dla studentów.

*workshop\_id* identyfikator warsztatu (klucz główny),

*conference\_day\_id* identyfikator dnia konferencji, w którym odbywa się warsztat

(klucz obcy),

*name*  nazwa warsztatu,

*place* miejsce, w którym odbywa się warsztat,

*start\_time* czas rozpoczęcia warsztatu,

*end\_time* czas zakończenia warsztatu,

*max\_participants* maksymalna ilość uczestników warsztatu,

*price* cena uczestnictwa w warsztacie,

CREATE TABLE workshop (

  workshop\_id integer NOT NULL CONSTRAINT workshop\_pk PRIMARY KEY identity,

  conference\_day\_id integer NOT NULL,

  name varchar(100) NOT NULL,

  place varchar(100),

  start\_time datetime,

  end\_time datetime ,

  max\_participants integer DEFAULT 1000,

  price money

);

## price

Tabela przechowuje informacje o cenie uczestnictwa danego dnia konferencji. Przewidziane są pola pozwalające uwzględnić rabat oraz ulgę dla studentów.

*price\_id* identyfikator pola (klucz główny),

*conference\_day\_id* identyfikator dnia konferencji (klucz obcy),

*student\_discount* zniżka studencka,

*discount* aktualna zniżka, za wcześniejszą rejestrację,

*value*  cena,

CREATE TABLE price (

  price\_id integer NOT NULL CONSTRAINT price\_pk PRIMARY KEY identity,

  conference\_day\_id integer NOT NULL,

  student\_discount decimal(2,2) DEFAULT 0,

  discount decimal(2,2) DEFAULT 0,

  value money NOT NULL

);

## Zainicjowanie bazy

Polecenie tworzące bazę danych przechowującą wyróżnione tabele, jednocześnie sprawdzające, czy dana baza już nie istnieje. Gdy baza istnieje, zostaje ona usunięta.

USE master

IF DB\_ID(N'conferences') is NOT NULL

DROP DATABASE conferences

CREATE DATABASE conferences

## Usunięcie tabel o korelujących nazwach

Przed stworzeniem nowych tabel należy usunąć wszystkie tabele w ramach danej bazy (u nas conferences), które posiadają przystającą nazwę do inicjowanych.

use conferences

IF OBJECT\_ID('client','U') IS NOT NULL

DROP TABLE client

IF OBJECT\_ID('reservation','U') IS NOT NULL

DROP TABLE reservation

IF OBJECT\_ID('payment','U') IS NOT NULL

DROP TABLE payment

IF OBJECT\_ID('conference\_attendee\_list','U') IS NOT NULL

DROP TABLE conference\_attendee\_list

IF OBJECT\_ID('conference','U') IS NOT NULL

DROP TABLE conference

IF OBJECT\_ID('workshop\_reservation','U') IS NOT NULL

DROP TABLE workshop\_reservation

IF OBJECT\_ID('attendee','U') IS NOT NULL

DROP TABLE attendee

IF OBJECT\_ID('workshop\_attendee\_list','U') IS NOT NULL

DROP TABLE workshop\_attendee\_list

IF OBJECT\_ID('workshop','U') IS NOT NULL

DROP TABLE workshop

IF OBJECT\_ID('conference\_day','U') IS NOT NULL

DROP TABLE conference\_day

IF OBJECT\_ID('price','U') IS NOT NULL

DROP TABLE price

## Wprowadzenie kluczy obcych

Należy skorelować ze sobą tabele. Deklaracje kluczy obcych poza inicjalizacją tabeli zapobiega błędom.

--client

--reservation

ALTER TABLE reservation

ADD FOREIGN KEY (client\_id) REFERENCES client(client\_id);

ALTER TABLE reservation

ADD FOREIGN KEY (conference\_day\_id) REFERENCES conference\_day(conference\_day\_id);

--payment

ALTER TABLE payment

ADD FOREIGN KEY (reservation\_id) REFERENCES reservation(reservation\_id);

--attendee

--conference\_attedee\_list

ALTER TABLE conference\_attendee\_list

ADD FOREIGN KEY (reservation\_id) REFERENCES reservation(reservation\_id);

ALTER TABLE conference\_attendee\_list

ADD FOREIGN KEY (attendee\_id) REFERENCES attendee(attendee\_id);

ALTER TABLE conference\_attendee\_list

ADD FOREIGN KEY (conference\_day\_id) REFERENCES conference\_day(conference\_day\_id);

--workshop\_attendee\_list

ALTER TABLE workshop\_attendee\_list

ADD FOREIGN KEY (attendee\_id) REFERENCES attendee(attendee\_id);

ALTER TABLE workshop\_attendee\_list

ADD FOREIGN KEY (workshop\_reservation\_id) REFERENCES workshop\_reservation(workshop\_reservation\_id);

--workshop\_reservation

ALTER TABLE workshop\_reservation

ADD FOREIGN KEY (workshop\_id) REFERENCES workshop(workshop\_id);

ALTER TABLE workshop\_reservation

ADD FOREIGN KEY (reservation\_id) REFERENCES reservation(reservation\_id);

--workshop

ALTER TABLE workshop

ADD FOREIGN KEY (conference\_day\_id) REFERENCES conference\_day(conference\_day\_id);

--conference

--conference\_day

ALTER TABLE conference\_day

ADD FOREIGN KEY (conference\_id) REFERENCES conference(conference\_id);

--price

ALTER TABLE price

ADD FOREIGN KEY (conference\_day\_id) REFERENCES conference\_day(conference\_day\_id);

# Funkcje

## conference\_days

Zwraca dni w trakcie których trwa podana konferencja o podanym ID.

CREATE FUNCTION conference\_days

   (

         @conference\_id         int

   )

   RETURNS TABLE

AS

   RETURN

         SELECT conference\_day\_id, date, max\_participants

         FROM conference\_day

         WHERE conference\_id = @conference\_id

## conference\_days\_cost

Zwraca koszt uczestnictwa w konferencji o podanym ID uwzględniając rabat czasowy.

CREATE FUNCTION conference\_days\_cost

   (

         @conference\_id int

   )

   RETURNS TABLE

AS

   RETURN

         SELECT price.conference\_day\_id, value \* (1 - discount) as [actual price],   
                             value \* (1 - student\_discount) as [actual price for students], value

         FROM price

         INNER JOIN conference\_day

         ON price.conference\_day\_id = conference\_day.conference\_day\_id

         WHERE conference\_id = @conference\_id

## workshop\_cost

Zwraca koszt warsztatu o podanym ID.

CREATE FUNCTION workshop\_cost

   (

         @workshop\_id int

   )

   RETURNS int

AS

BEGIN

   IF NOT EXISTS (SELECT \* FROM workshop WHERE workshop\_id = @workshop\_id)

   RETURN CAST ('Workshop with given id do not exist' AS int)

   RETURN (SELECT price FROM workshop WHERE workshop\_id = @workshop\_id)

END

## places\_reserved\_on\_conference

Zwraca liczbę miejsc zarezerwowanych w ramach jednej konferencji o podanym ID.

CREATE FUNCTION places\_reserved\_on\_conference

   (

         @conference\_id int

   )

   RETURNS TABLE

AS

   RETURN

                     SELECT cd.conference\_day\_id, date, sum(places\_reserved) as [reserved places]

      FROM conference\_day as cd

                      LEFT OUTER JOIN reservation as r

         ON cd.conference\_day\_id = r.conference\_day\_id AND isCancelled = 0

         WHERE conference\_id = @conference\_id

           GROUP BY conference\_id, cd.conference\_day\_id, date

## places\_available\_on\_conference

Zwraca liczbę dostępnych miejsc na konferencję o podanym ID.

CREATE FUNCTION places\_available\_on\_conference

   (

         @conference\_id int

   )

   RETURNS TABLE

AS

   RETURN

           SELECT cd.conference\_day\_id, date,  
 max\_participants - sum(places\_reserved) as [available places]

     FROM conference\_day as cd

           LEFT OUTER JOIN reservation as r

           ON cd.conference\_day\_id = r.conference\_day\_id

           WHERE conference\_id = @conference\_id

           GROUP BY conference\_id, cd.conference\_day\_id, date, max\_participants

## places\_reserved\_on\_workshop

Zwraca liczbę zarezerwowanych miejsc na warsztat o podanym ID.

CREATE FUNCTION places\_reserved\_on\_workshop

   (

         @workshop\_id int

   )

   RETURNS int

AS

BEGIN

   RETURN  (SELECT sum(places\_reserved) as [reserved places]

                FROM workshop\_reservation

                WHERE workshop\_id = @workshop\_id AND isCancelled = 0

         GROUP BY workshop\_id)

END

## places\_available\_on\_workshop

Zwraca liczbę dostępnych miejsc na warsztat o podanym ID.

CREATE FUNCTION places\_available\_on\_workshop

   (

         @workshop\_id int

   )

   RETURNS int

AS

BEGIN

   RETURN  (SELECT max\_participants - sum(places\_reserved) as [available places]

                FROM workshop as w

                LEFT OUTER JOIN workshop\_reservation as wr

                ON w.workshop\_id = wr.workshop\_id

                WHERE w.workshop\_id = @workshop\_id AND isCancelled = 0

         GROUP BY w.workshop\_id, max\_participants)

END

## conference\_attendees

Zwraca listę uczestników konferencji o podanym ID, bez uwzględniania podziału na dni.

CREATE FUNCTION conference\_attendees

   (

         @conference\_id      int

   )

   RETURNS TABLE

AS

   RETURN

         SELECT c\_d.conference\_day\_id, date, first\_name, last\_name, company\_name

                FROM attendee AS a

                INNER JOIN conference\_attendee\_list AS c\_a\_l

                ON a.attendee\_id = c\_a\_l.attendee\_id

                INNER JOIN reservation AS r

                ON r.reservation\_id = c\_a\_l.reservation\_id AND isCancelled = 0

                INNER JOIN client AS c

                ON c.client\_id = r.client\_id

                INNER JOIN conference\_day AS c\_d

                ON c\_d.conference\_day\_id = r.conference\_day\_id

                WHERE c\_d.conference\_id = @conference\_id

## conference\_day\_attendees

Zwraca listę uczestników zapisanych na dany dzień konferencji o podanym ID.

CREATE FUNCTION conference\_day\_attendees

   (

   @conference\_day\_id  int

   )

   RETURNS TABLE

AS

   RETURN (SELECT \* FROM attendee

                WHERE attendee\_id IN (SELECT attendee\_id FROM conference\_attendee\_list

                                                   WHERE conference\_day\_id = @conference\_day\_id))

## workshop\_attendees

Zwraca listę uczestników warsztatu o podanym ID.

CREATE FUNCTION workshop\_attendees

   (

         @workshop\_id      int

   )

   RETURNS TABLE

AS

   RETURN

         SELECT first\_name, last\_name, company\_name

                FROM attendee AS a

                INNER JOIN workshop\_attendee\_list AS w\_a\_l

                ON w\_a\_l.attendee\_id = a.attendee\_id

                INNER JOIN workshop\_reservation AS w\_r

                ON w\_r.workshop\_reservation\_id = w\_a\_l.workshop\_reservation\_id AND  
 w\_r.isCancelled = 0

                INNER JOIN reservation AS r

                ON r.reservation\_id = w\_r.reservation\_id AND r.isCancelled = 0

                INNER JOIN client AS c

                ON c.client\_id = r.client\_id

                WHERE w\_r.workshop\_id = @workshop\_id

## client\_payments

Zwraca płatności klienta o podanym ID.

CREATE FUNCTION client\_payments

   (

   @client\_id int

   )

   RETURNS int

AS

BEGIN

   IF NOT EXISTS (SELECT \* FROM client WHERE client\_id = @client\_id)

   RETURN CAST('Client does not exist' AS int)

   RETURN (SELECT SUM(value) FROM payment

   WHERE reservation\_id IN (

         SELECT reservation\_id FROM reservation

         WHERE client\_id = @client\_id AND isCancelled = 0))

END

## numer\_of\_students

Zwraca listę studentów zgłoszonych przez klienta na dany dzień konferencji.

CREATE FUNCTION number\_of\_students

   (

         @client\_id             int,

         @conference\_day\_id  int

   )

RETURNS int

AS

BEGIN

   DECLARE @number\_of\_students int;

   SET @number\_of\_students  = (SELECT sum(CASE WHEN student\_number IS NOT NULL  
 THEN 1 ELSE 0 END)

                                            FROM attendee AS a

                                            INNER JOIN conference\_attendee\_list AS c\_a\_l

                                            ON c\_a\_l.attendee\_id = a.attendee\_id

                                            WHERE c\_a\_l.conference\_day\_id = @conference\_day\_id)

   RETURN @number\_of\_students

END

# Procedury

## add\_conference

Wprowadza nową konferencje.

CREATE PROCEDURE add\_conference

@name varchar(150),

@address varchar(100) = NULL,

@city varchar(20) = NULL

AS

BEGIN

IF EXISTS (SELECT \* FROM conference WHERE name = @name AND address = @address AND city = @city)

BEGIN

THROW 50000, 'Conference with given data already exists', 1

END

INSERT INTO conference (name, address, city)

VALUES (@name, @address, @city)

END

## add\_client

Wprowadza nowego klienta.

CREATE PROCEDURE add\_client

@company\_name varchar(50) = NULL,

@contact\_name varchar(50),

@mail varchar(50),

@address varchar(100) = NULL,

@phone varchar(12)

AS

BEGIN

IF EXISTS (SELECT \* FROM client WHERE company\_name = @company\_name AND contact\_name = @contact\_name AND address = @address)

BEGIN

THROW 50000, 'Client with given data already exists', 1

END

INSERT INTO client(company\_name, contact\_name, [e-mail], address, phone)

VALUES(@company\_name, @contact\_name, @mail, @address, @phone)

END

## add\_payment

Wprowadza płatność.

CREATE PROCEDURE new\_payment

   @reservation\_id         int,

   @payment\_date    date,

   @value                  money

AS

BEGIN

   IF NOT EXISTS(SELECT \* FROM reservation WHERE reservation\_id = @reservation\_id)

   BEGIN

         THROW 50000,'Reservation does not exist',1

   END

IF EXISTS(SELECT \* FROM reservation WHERE reservation\_id = @reservation\_id   
 AND isCancelled = 1)

BEGIN

THROW 50000,'Reservation has been cancelled',1

END

   DECLARE @actual\_date as date

   SET @actual\_date = getdate()

   IF (@actual\_date < @payment\_date)

   BEGIN

         THROW 50000,'Cannot add future payments',1

   END

   INSERT INTO payment (reservation\_id,payment\_date,value)

   VALUES (@reservation\_id, @payment\_date, @value)

END

## add\_conference\_day

Wprowadza dzień konferencje.

CREATE PROCEDURE add\_conference\_day

   @conference\_id     int,

   @date               datetime,

   @max\_participants  int = NULL

AS

BEGIN

   IF NOT EXISTS (SELECT \* FROM conference

                  WHERE conference\_id = @conference\_id)

   BEGIN

         THROW 50000, 'Conference with given id does not exist ', 1

   END

   IF EXISTS (SELECT \* FROM conference\_day

                  WHERE conference\_id = @conference\_id AND date = @date)

   BEGIN

         THROW 50000, 'Conference day with given id and date already exist', 1

   END

   INSERT INTO conference\_day (conference\_id, date, max\_participants)

   VALUES (@conference\_id, @date, @max\_participants)

END

## add\_workshop

Wprowadza warsztat.

CREATE PROCEDURE add\_workshop

         @conference\_day\_id   int,

         @name                  varchar(100),

         @place                 varchar(100) = NULL,

         @start\_time             datetime = NULL,

         @end\_time              datetime = NULL,

         @max\_participants    int = NULL,

         @price                 money = NULL

AS

BEGIN

   IF NOT EXISTS (SELECT \* FROM conference\_day

                                   WHERE conference\_day\_id = @conference\_day\_id)

   BEGIN

         THROW 50000, 'Conference with given conference\_day\_id not exist', 1

   END

   IF EXISTS (SELECT \* FROM workshop

                     WHERE conference\_day\_id = @conference\_day\_id AND name = @name)

   BEGIN

         THROW 50000, 'Workshop with given name and conference\_day\_id already exist', 1

  END

   INSERT INTO workshop (conference\_day\_id, name, place, start\_time, end\_time,   
                                                       max\_participants, price)

   VALUES (@conference\_day\_id, @name, @place, @start\_time, @end\_time,   
                             @max\_participants, @price)

END

## add\_price

Wprowadza cenę za dzień konferencje.

CREATE PROCEDURE add\_price

   @conference\_day\_id   int,

   @value                 money,

   @discount              decimal(2,2) = NULL,

   @student\_discount    decimal(2,2) = NULL

AS

BEGIN

   IF NOT EXISTS (SELECT \* FROM conference\_day

                        WHERE conference\_day\_id = @conference\_day\_id)

   BEGIN

         THROW 50000, 'Conference with given conference\_day\_id not exist', 1

   END

   INSERT INTO price (conference\_day\_id, value, discount, student\_discount)

   VALUES (@conference\_day\_id, @value, @discount, @student\_discount)

END

## add\_price\_in\_range

Dodawanie jednakowej ceny na każdy dzień konferencji w podanym zakresie.

CREATE PROCEDURE add\_price\_in\_range

   @conference\_id          int,

   @value                 money,

   @start\_date             date,

   @end\_date              date,

   @discount              decimal(2,2) = NULL,

   @student\_discount    decimal(2,2) = NULL

AS

BEGIN

   IF NOT EXISTS (SELECT \* FROM conference

                  WHERE conference\_id = @conference\_id)

   BEGIN

         THROW 50000, 'Conference with given id does not exist ', 1

   END

   IF @end\_date > @start\_date

   BEGIN

         THROW 50000, 'Incorrect data: endDate must be after startDate', 1

   END

   IF CAST(@start\_date AS DATE) < (SELECT MIN(date) FROM conference\_day

                              WHERE conference\_id = @conference\_id)

   BEGIN

         THROW 50000, 'Incorrect data: startDate can not be before first conference date', 1

   END

   IF CAST(@end\_date AS DATE) > (SELECT MAX(date) FROM conference\_day

                                               WHERE conference\_id = @conference\_id)

   BEGIN

         THROW 50000, 'Incorrect data: startDate can not be before first conference date', 1

   END

   IF DATEDIFF(DAY, @start\_date, @end\_date) != (SELECT COUNT(date) FROM  
 conference\_day

                                                                                      WHERE conference\_id = @conference\_id)

   BEGIN

         THROW 50000, 'Incorrect data: invalid date interval', 1

   END

   DECLARE @actual\_date as date

   SET @actual\_date = @start\_date

   WHILE @actual\_date <= @end\_date

         BEGIN

                DECLARE @day\_id as int

                SET @day\_id = (SELECT conference\_day\_id FROM conference\_day

                                                WHERE conference\_id = @conference\_id AND date = @actual\_date)

                INSERT INTO price(conference\_day\_id, value, discount, student\_discount)

                VALUES (@day\_id, @value, @discount, @student\_discount)

                SET @actual\_date = DATEADD(DAY, 1, @actual\_date)

         END

END

## add\_attendee

Dodawanie uczestnika do bazy danych.

CREATE PROCEDURE add\_attendee

   @first\_name varchar(15),

   @last\_name varchar(15),

   @student\_number varchar(10) = NULL

AS

BEGIN

   IF EXISTS (SELECT \* FROM attendee WHERE student\_number = @student\_number)

   BEGIN

         THROW 50000,'This student number is already in database', 1

   END

   INSERT INTO attendee(first\_name, last\_name, student\_number)

   VALUES (@first\_name, @last\_name, @student\_number)

END

## add\_attendee\_on\_conference

Dodawanie uczestnika na dzień konferencji.

CREATE PROCEDURE add\_attendee\_on\_conference

   @conference\_day\_id int,

   @attendee\_id int

AS

BEGIN

   IF NOT EXISTS(SELECT \* FROM conference\_day WHERE conference\_day\_id = @conference\_day\_id)

   BEGIN

         THROW 50000, 'Conference day with this ID does not exist',1

   END

   IF NOT EXISTS(SELECT \* FROM attendee WHERE attendee\_id = @attendee\_id)

   BEGIN

         THROW 50000, 'Attendee with this ID does not exist',1

   END

IF EXISTS(SELECT \* FROM reservation WHERE conference\_day\_id = @conference\_day\_id  
 AND isCancelled = 1)

BEGIN

THROW 50000, 'Reservation has been cancelled',1

END

   DECLARE @reservation\_id as int

   SET @reservation\_id = (SELECT reservation\_id FROM reservation  
 WHERE conference\_day\_id = @conference\_day\_id)

   DECLARE @participants as int

   SET @participants = (SELECT count(\*) from conference\_attendee\_list   
 WHERE conference\_day\_id = @conference\_day\_id)

   DECLARE @maxparticipants as int

   SET @maxparticipants = (SELECT max\_participants from conference\_day   
 WHERE conference\_day\_id = @conference\_day\_id)

   IF (@participants = @maxparticipants)

   BEGIN

         THROW 50000, 'Conference day is full',1

   END

   INSERT INTO conference\_attendee\_list (reservation\_id, attendee\_id, conference\_day\_id)

   VALUES (@reservation\_id, @attendee\_id, @conference\_day\_id)

END

## add\_attendee\_on\_workshop

Dodawanie uczestnika do warsztatu.

CREATE PROCEDURE add\_attendee\_on\_workshop

   @workshop\_id int,

   @attendee\_id int

AS

BEGIN

   IF NOT EXISTS(SELECT \* FROM workshop WHERE workshop\_id = @workshop\_id)

   BEGIN

         THROW 50000, 'Workshop with this ID does not exist',1

   END

   IF NOT EXISTS(SELECT \* FROM attendee WHERE attendee\_id = @attendee\_id)

   BEGIN

         THROW 50000, 'Attendee with this ID does not exist',1

   END

IF EXISTS(SELECT \* FROM workshop\_reservation   
 WHERE workshop\_id = @workshop\_id AND isCancelled = 1)

BEGIN

THROW 50000, 'Reservation has been cancelled',1

END

   DECLARE @workshop\_reservation\_id as int

   SET @workshop\_reservation\_id = (SELECT workshop\_reservation\_id   
 FROM workshop\_reservation   
 WHERE workshop\_id = @workshop\_id)

   DECLARE @participants as int

   SET @participants = (SELECT count(\*) from workshop\_attendee\_list   
 WHERE workshop\_reservation\_id = @workshop\_id)

   DECLARE @maxparticipants as int

   SET @maxparticipants = (SELECT max\_participants from workshop   
 WHERE workshop\_id = @workshop\_id)

   IF (@participants = @maxparticipants)

   BEGIN

         THROW 50000, 'Workshop day is full',1

   END

   INSERT INTO workshop\_attendee\_list (attendee\_id, workshop\_reservation\_id)

   VALUES (@attendee\_id, @workshop\_reservation\_id)

END

## new\_conference\_reservation

Wprowadza nową rezerwację na konferencję.

CREATE PROCEDURE new\_conference\_reservation

   @client\_id              int,

   @conference\_day\_id  int,

   @reservation\_date   date,

   @places\_reserved int = NULL

AS

BEGIN

   IF NOT EXISTS(SELECT \* FROM client WHERE client\_id = @client\_id)

   BEGIN

         THROW 50000,'Client does not exist',1

   END

   IF NOT EXISTS(SELECT \* FROM conference\_day WHERE conference\_day\_id = @conference\_day\_id)

   BEGIN

         THROW  50000,'Reservation day does not exist',1

   END

   DECLARE @actual\_date as date

   IF (@reservation\_date < @actual\_date)

   BEGIN

         THROW 50000,'Cannot travel in time',1

   END

   INSERT INTO reservation (client\_id, conference\_day\_id, reservation\_date, places\_reserved)

   VALUES (@client\_id, @conference\_day\_id, @reservation\_date, @places\_reserved)

END

## new\_workshop\_reservation

Wprowadza rezerwację na warsztat.

CREATE PROCEDURE new\_workshop\_reservation

@workshop\_id int,

@reservation\_id int,

@places\_reserved int = NULL

AS

BEGIN

IF NOT EXISTS(SELECT \* FROM workshop WHERE workshop\_id = @workshop\_id)

BEGIN

THROW 50000,'Workshop does not exist',1

END

IF NOT EXISTS(SELECT \* FROM reservation WHERE reservation\_id = @reservation\_id)

BEGIN

THROW 50000,'Reservation does not exist',1

END

DECLARE @maxparticipants as int

SET @maxparticipants = (SELECT max\_participants from workshop WHERE workshop\_id = @workshop\_id)

IF (@maxparticipants < @places\_reserved)

BEGIN

THROW 50000,'Cannot reserve as much places on this workshop',1

END

INSERT INTO workshop\_reservation (workshop\_id, places\_reserved)

VALUES (@workshop\_id, @places\_reserved)

END

## update\_conference\_participants\_limit

Aktualizuje maksymalną liczbę uczestników na dzień konferencji.

CREATE PROCEDURE update\_conference\_participants\_limit

   @conference\_day\_id      int,

   @max\_participants   int

AS

BEGIN

   IF NOT EXISTS (SELECT \* FROM conference\_day

                        WHERE conference\_day\_id = @conference\_day\_id)

   BEGIN

         THROW 50000, 'Conference with given conference\_day\_id does not exist ', 1

   END

   IF @max\_participants < (SELECT SUM(places\_reserved) FROM reservation

                                          WHERE conference\_day\_id = @conference\_day\_id)

   BEGIN

         THROW 50000, 'New limit is greater than number of already reserved places', 1

   END

   UPDATE conference\_day

   SET max\_participants = @max\_participants

   WHERE conference\_day\_id = @conference\_day\_id

END

## update\_workshop\_participants\_limit

Aktualizuje maksymalną liczbę uczestników na warsztat.

CREATE PROCEDURE update\_workshop\_participants\_limit

   @workshop\_id     int,

   @max\_participants    int

AS

BEGIN

   IF NOT EXISTS (SELECT \* FROM workshop

                        WHERE workshop\_id = @workshop\_id)

   BEGIN

         THROW 50000, 'Workshop with given id does not exist ', 1

   END

   IF @max\_participants < (SELECT SUM(places\_reserved) FROM workshop\_reservation

                                          WHERE workshop\_id = @workshop\_id)

   BEGIN

         THROW 50000, 'New limit is greater than number of already reserved places', 1

   END

   UPDATE workshop

   SET max\_participants = @max\_participants

   WHERE workshop\_id = @workshop\_id

END

## cancel\_reservation

Anulowanie rezerwacji na konferencję.

CREATE PROCEDURE cancel\_reservation

   @reservation\_id         int

AS

BEGIN

   IF NOT EXISTS(SELECT \* FROM reservation WHERE reservation\_id = @reservation\_id)

   BEGIN

         THROW 50000,'Reservation does not exist',1

   END

   DECLARE @actual\_state as bit

   SET @actual\_state = (SELECT isCancelled FROM reservation WHERE   
 reservation\_id = @reservation\_id)

   IF (@actual\_state = 1)

   BEGIN

         THROW 50000,'Reservation is already cancelled',1

   END

   UPDATE reservation SET isCancelled = 1 WHERE reservation\_id = @reservation\_id

END

## cancel\_workshop

Anulowanie rezerwacji na warsztat.

CREATE PROCEDURE cancel\_workshop

   @workshop\_reservation\_id      int

AS

BEGIN

   IF NOT EXISTS(SELECT \* FROM workshop\_reservation WHERE workshop\_reservation\_id = @workshop\_reservation\_id)

   BEGIN

     THROW 50000,'Reservation does not exist',1

   END

   DECLARE @actual\_state as bit

   SET @actual\_state = (SELECT isCancelled FROM workshop\_reservation WHERE workshop\_reservation\_id = @workshop\_reservation\_id)

   IF (@actual\_state = 1)

   BEGIN

     THROW 50000,'Reservation is already cancelled',1

   END

   UPDATE workshop\_reservation SET isCancelled = 1 WHERE workshop\_reservation\_id = @workshop\_reservation\_id

END

\_id, attendee\_id, conference\_day\_id)

   VALUES (@reservation\_id, @attendee\_id, @conference\_day\_id)

END

## cancel\_all\_unpaid\_reservations

Anulowanie nieopłaconych rezerwacji.

CREATE PROCEDURE cancel\_unpaid\_reservations

AS

BEGIN

   WHILE (EXISTS(SELECT \* FROM reservation as r

                        WHERE reservation\_id NOT IN

                              (SELECT reservation\_id FROM payment)

                        AND r.isCancelled = 0))

   BEGIN

         DECLARE @res\_to\_cancel as int

         SET @res\_to\_cancel = (SELECT min(r.reservation\_id) FROM reservation as r

                                            WHERE reservation\_id NOT IN

                                                  (SELECT reservation\_id FROM payment)

                                            AND r.isCancelled = 0)

         UPDATE reservation SET isCancelled = 1 WHERE reservation\_id = @res\_to\_cancel

   END

END

## cancel\_unpaid\_reservations\_in\_time

Anulowanie nieopłaconych rezerwacji tydzień przed realizacją.

CREATE PROCEDURE cancel\_unpaid\_reservations\_in\_time

AS

BEGIN

   WHILE (EXISTS(SELECT \* FROM reservation as r

                        WHERE reservation\_id NOT IN

                              (SELECT reservation\_id FROM payment)

                        AND r.isCancelled = 0

                        AND (day(getdate()) - day(r.reservation\_date)) > 7))

   BEGIN

         DECLARE @res\_to\_cancel as int

         SET @res\_to\_cancel = (SELECT min(r.reservation\_id) FROM reservation as r

                                            WHERE reservation\_id NOT IN

                                                  (SELECT reservation\_id FROM payment)

                                            AND r.isCancelled = 0

                                            AND (day(getdate()) - day(r.reservation\_date)) > 7)

         UPDATE reservation SET isCancelled = 1 WHERE reservation\_id = @res\_to\_cancel

   END

END

# Widoki

## most\_active\_clients

Wyświetla tabelę klientów ze względu na malejącą ilość rezerwacji.

CREATE VIEW most\_active\_client

AS

SELECT count(r.reservation\_id) as 'Number of reservations', r.client\_id, min(c.contact\_name) as 'Contact name', min(c.phone) as 'Phone'

FROM reservation as r

INNER JOIN client as c ON c.client\_id = r.client\_id

GROUP BY r.client\_id

## available\_conferences

Wyświetla dostępne do zapisania się rezerwacje.

CREATE VIEW available\_conferences

AS

   SELECT \* FROM conference as c

   WHERE c.conference\_id IN

         (SELECT cd.conference\_id FROM conference\_day as cd

         WHERE date > getdate() AND --tylko przyszłe

         max\_participants >

                (SELECT count(\*) FROM conference\_attendee\_list as cal

                WHERE cal.conference\_day\_id = cd.conference\_day\_id) --wolne miejsca

                AND cd.conference\_day\_id IN

                       (SELECT r.conference\_day\_id FROM reservation as r

                       WHERE isCancelled = 0)) --nie anulowane

## available\_workshops

Wyświetla dostępne do zapisania się warsztaty.

CREATE VIEW available\_workshops

AS

   SELECT \* FROM workshop as w

   WHERE workshop\_id IN

         (SELECT wr.workshop\_id FROM workshop\_reservation as wr

         WHERE  isCancelled = 0 AND

         max\_participants >

                       (SELECT count(\*) FROM workshop\_attendee\_list as wal

                       WHERE wal.workshop\_reservation\_id = wr.workshop\_reservation\_id))

   AND start\_time > getdate()

## clients\_with\_payment\_deficit

Wyświetla klientów z nieuregulowanymi płatnościami.

CREATE VIEW clients\_with\_payment\_deficit

AS

   SELECT client\_id FROM reservation as r

   LEFT OUTER JOIN payment as p ON r.reservation\_id = p.reservation\_id

   INNER JOIN workshop\_reservation as wr ON wr.reservation\_id = r.reservation\_id

   INNER JOIN workshop as w ON w.workshop\_id = wr.workshop\_id

   INNER JOIN conference\_day as cd ON cd.conference\_day\_id = r.conference\_day\_id

   INNER JOIN price as c ON c.conference\_day\_id = cd.conference\_day\_id

   GROUP BY r.client\_id

   HAVING sum(p.value) < (sum(w.price) + sum(c.value))

## cancelled\_reservations

Wyświetla anulowane rezerwacje.

CREATE VIEW cancelled\_reservations

AS

SELECT \* FROM reservation

WHERE isCancelled = 1

## cancelled\_workshop\_reservations

Wyświetla anulowane rezerwacje na warsztaty.

CREATE VIEW cancelled\_workshop\_reservations

AS

SELECT \* FROM workshop\_reservation

WHERE isCancelled = 1

## clients\_to\_call

Wyświetla klientów, którzy nie uzupełnili list uczestników 2 tygodnie przed rozpoczęciem konferencji, do których trzeba zadzwonić.

CREATE VIEW clients\_to\_call

AS

SELECT client\_id, phone FROM client

WHERE client\_id IN

(SELECT client\_id FROM reservation as r

WHERE r.conference\_day\_id IN

(SELECT cd.conference\_day\_id FROM conference\_day as cd

WHERE datediff(day, cd.date, getdate()) < 14)

AND

((SELECT distinct count(\*) FROM conference\_attendee\_list as cal

WHERE cal.reservation\_id = r.reservation\_id) +

(SELECT sum(places\_reserved) FROM workshop\_reservation as wal

WHERE wal.reservation\_id = r.reservation\_id)) = r.places\_reserved)

## monthly\_income

Wyświetla dochód brutto ze względu na miesiące.

CREATE VIEW monthly\_income

AS

SELECT YEAR(payment\_date) as year, MONTH(payment\_date) as month, sum(value) as income

FROM payment

GROUP BY YEAR(payment\_date), MONTH(payment\_date)

## company\_clients

Wyświetla klientów będących firmami.

CREATE VIEW company\_clients

AS

SELECT \* FROM client

WHERE company\_name IS NOT NULL

## attendee\_stats

Wyświetla ilość wydarzeń na których zapisany był dany uczestnik.

CREATE VIEW attendee\_stats

AS

SELECT a.attendee\_id, ((SELECT count(\*) FROM

conference\_attendee\_list as cal

WHERE cal.attendee\_id = a.attendee\_id) +

(SELECT count(\*) FROM

workshop\_attendee\_list as wal

WHERE wal.attendee\_id = a.attendee\_id)) as 'sum'

FROM attendee as a

## unpaid\_reservations

Wyświetla nieopłacone rezerwacje.

CREATE VIEW unpaid\_reservations

AS

   SELECT r.reservation\_id, reservation\_date, c\_d.date as [conference date], company\_name,  
 contact\_name, [e-mail], phone, places\_reserved

   FROM reservation AS r

   INNER JOIN client AS c

   ON c.client\_id = r.client\_id

   INNER JOIN conference\_day AS c\_d

   ON c\_d.conference\_day\_id = r.conference\_day\_id

   LEFT OUTER JOIN payment AS p

   ON p.reservation\_id = r.reservation\_id

   WHERE payment\_id IS NULL

## clients\_with\_less\_attendees\_than\_reserved

Wyświetla klientów, którzy mają więcej zarezerwowanych miejsc na konferencję, niż   
przypisanych do niech uczestników.

CREATE VIEW clients\_with\_less\_attendees\_than\_reserved

AS

   SELECT c.client\_id, company\_name, r.conference\_day\_id, places\_reserved, places\_reserved - count(a.attendee\_id) as [missing attendees]

   FROM client AS c

   LEFT OUTER JOIN reservation AS r

   ON r.client\_id = c.client\_id

   LEFT OUTER JOIN conference\_attendee\_list AS c\_a\_l

   ON c\_a\_l.reservation\_id = r.reservation\_id

   LEFT OUTER JOIN attendee AS a

   ON a.attendee\_id = c\_a\_l.attendee\_id

   GROUP BY c.client\_id, company\_name, r.conference\_day\_id, r.reservation\_id, places\_reserved

   HAVING  places\_reserved - count(a.attendee\_id) > 0

## clients\_with\_less\_attendees\_than\_reserved\_for\_workshop

Wyświetla klientów, którzy mają więcej zarezerwowanych miejsc na konferencję, niż   
przypisanych do niech uczestników.

CREATE VIEW clients\_with\_less\_attendees\_than\_reserved\_for\_workshop

AS

   SELECT c.client\_id, company\_name, r.conference\_day\_id, workshop\_id, w\_r.places\_reserved, w\_r.places\_reserved - count(a.attendee\_id) as [missing attendees]

   FROM client AS c

   LEFT OUTER JOIN reservation AS r

   ON r.client\_id = c.client\_id

   LEFT OUTER JOIN workshop\_reservation as w\_r

   ON w\_r.reservation\_id = r.reservation\_id

   LEFT OUTER JOIN workshop\_attendee\_list AS w\_a\_l

   ON w\_a\_l.workshop\_reservation\_id = w\_r.workshop\_reservation\_id

   LEFT OUTER JOIN attendee AS a

   ON a.attendee\_id = w\_a\_l.attendee\_id

   GROUP BY c.client\_id, company\_name, r.conference\_day\_id, workshop\_id, r.reservation\_id, w\_r.places\_reserved

   HAVING  w\_r.places\_reserved - count(a.attendee\_id) > 0

# Triggery

## checking\_number\_of\_reserved\_places

Sprawdza, czy ilość zarezerwowanych miejsc na dzień konferencji nie przekracza ilości dostępnych miejsc.

CREATE TRIGGER checking\_number\_of\_reserved\_places

   ON reservation

   AFTER INSERT, UPDATE

AS

BEGIN

   IF EXISTS (SELECT \* FROM inserted

                  WHERE dbo.places\_available\_on\_conference\_day(inserted.conference\_day\_id) < 0)

         BEGIN

         THROW 50000, 'This conference has more reserved than available places', 1

         END

END

## checking\_number\_of\_reserved\_places\_workshop

Sprawdza, czy ilość zarezerwowanych miejsc na warsztat nie przekracza ilości dostępnych miejsc.

CREATE TRIGGER checking\_number\_of\_reserved\_places\_workshop

   ON workshop\_reservation

   AFTER INSERT, UPDATE

AS

BEGIN

   IF EXISTS (SELECT \* FROM inserted

                  WHERE dbo.places\_available\_on\_workshop(inserted.workshop\_id) < 0)

         BEGIN

         THROW 50000, 'This workshop has more reserved than available places', 1

         END

END

## too\_many\_attendees\_for\_conference

Sprawdza, czy ilość zgłoszonych uczestników konferencji nie przekracza ilości zarezerwowanych miejsc .

CREATE TRIGGER too\_many\_attendees\_for\_conference

   ON conference\_attendee\_list

   AFTER INSERT, UPDATE

AS

BEGIN

   IF (SELECT count(\*) FROM conference\_attendee\_list WHERE reservation\_id =   
 (SELECT reservation\_id FROM inserted)) >   
 (SELECT MIN(places\_reserved) FROM reservation WHERE reservation\_id =  
 (SELECT reservation\_id FROM inserted))

   BEGIN

         THROW 50000, 'Number of attendees can not be greater than number of reserved places', 1

   END

END

## too\_many\_attendees\_for\_workshop

Sprawdza, czy ilość zgłoszonych uczestników warsztatu nie przekracza ilości zarezerwowanych miejsc

CREATE TRIGGER too\_many\_attendees\_for\_workshop

   ON workshop\_attendee\_list

   AFTER INSERT, UPDATE

AS

BEGIN

   IF

   (SELECT count(\*) FROM workshop\_attendee\_list WHERE workshop\_reservation\_id =

   (SELECT workshop\_reservation\_id FROM inserted)) >   
 (SELECT MIN(places\_reserved) FROM workshop\_reservation   
 WHERE workshop\_reservation\_id = (SELECT workshop\_reservation\_id FROM inserted))

   BEGIN

         THROW 50000, 'Number of attendees can not be greater than number of reserved places', 1

   END

END

## max\_participants\_for\_workshop

Sprawdza, czy ilość zarezerwowanych miejsc na warsztat nie jest większa niż na całej konferencji.

CREATE TRIGGER max\_participants\_for\_workshop

   ON workshop

   AFTER INSERT, UPDATE

AS

BEGIN

   IF (SELECT max\_participants FROM inserted) >  
 (SELECT max\_participants FROM conference\_day WHERE conference\_day\_id =   
 (SELECT conference\_day\_id FROM inserted))

   BEGIN

         THROW 50000, 'Workshop can not have more available places than conference', 1

   END

END

## checking\_number\_of\_reserved\_places\_for\_individual\_client

Sprawdza ilość zarezerwowanych miejsc przez klienta indywidualnego.

CREATE TRIGGER checking\_number\_of\_reserved\_places\_for\_individual\_client

   ON reservation

   AFTER INSERT

AS

BEGIN

   IF (SELECT company\_name FROM client WHERE client\_id = (SELECT client\_id FROM inserted)) IS NULL AND (SELECT places\_reserved FROM inserted) != 1

         BEGIN

         THROW 50000, 'Individual client can only reserve one place', 1

         END

END

## checking\_date

Sprawdza czy nie są rezerwowane miejsca na konferencję, która już się odbyła.

CREATE TRIGGER checking\_date

   ON reservation

   AFTER INSERT, UPDATE

AS

BEGIN

   IF (SELECT date FROM conference\_day WHERE conference\_day\_id = (SELECT conference\_day\_id FROM inserted)) > getDate()

         BEGIN

         THROW 50000, 'Reservation for a conference that took place can not be done', 1

         END

END

## checking\_date\_for\_workshop

Sprawdza czy nie są rezerwowane miejsca na warsztat, który już się

CREATE TRIGGER checking\_date\_for\_workshop

   ON workshop\_reservation

   AFTER INSERT, UPDATE

AS

BEGIN

   IF (SELECT end\_time FROM workshop WHERE workshop\_id = (SELECT workshop\_id FROM inserted)) > getDate()

         BEGIN

         THROW 50000, 'Reservation for a workshop that took place can not be done', 1

         END

END

## reservation\_cancel

Anuluje rezerwacje na warsztaty przy anulowaniu rezerwacji na konferencję.

CREATE TRIGGER reservation\_cancel

ON reservation

AFTER UPDATE

AS

BEGIN

IF (SELECT isCancelled FROM UPDATED) = 1

UPDATE workshop\_reservation SET isCancelled = 1

WHERE reservation\_id = (SELECT reservation\_id FROM UPDATED)

END

# Indeksy

Zdefiniowaliśmy indeksy po kluczach obcych do innych tabel. Pominęto indeksy dla workshop\_attendee\_list i conference\_attendee\_list, ponieważ istnieje możliwość usunięcia tych pól, więc indeksowanie nie ma sensu.

CREATE INDEX conference\_c\_days\_id\_index ON conference\_day(conference\_id)

CREATE INDEX price\_cd\_day\_id\_index ON price(conference\_day\_id)

CREATE INDEX workshops\_day\_cd\_id\_index ON workshop(conference\_day\_id)

CREATE INDEX workshop\_reservations\_w\_id\_index ON workshop\_reservation(workshop\_id)

CREATE INDEX workshop\_reservations\_r\_id\_index ON workshop\_reservation(reservation\_id)

CREATE INDEX reservations\_cd\_day\_id\_index ON reservation(conference\_day\_id)

CREATE INDEX reservations\_cl\_id\_index ON reservation(client\_id)

CREATE INDEX payments\_r\_id\_index ON payment(reservation\_id)

# Role w systemie

Podział na role jest niezbędne do sprawnego zarządzania systemem

## Administrator systemu

Nieograniczony dostęp do bazy danych, możliwość modyfikacji widoków, funkcji i triggerów.

## Pracownik firmy

Dostęp do wszystkich funkcji i widoków, możliwość edytowania zawartości tabel.

## Klient

Dostęp do funkcji powiązanych z tworzeniem nowych i anulowaniem rezerwacji, zmiana parametru places\_reserved, a także do widoków, które prezentują dostępne miejsca na warsztat/conferencje.

## Uczestnik

Może zobaczyć na jakie konferencję i warsztaty jest zapisany.

# Analiza wymagań

## Klient

* Podgląd kalendarza wydarzeń
* Rejestracja na wybrane dni konferencji
* Edycja danych klienta
* Wyświetlanie danych o płatnościach
* Rezygnacja z miejsca na konferencji

## Klient (firma)

* Podgląd kalendarza wydarzeń
* Rezerwacja miejsc na konferencji
* Rejestracja uczestników na wybrane dni konferencji
* Uzupełnienie danych uczestnika
* Edycja danych klienta
* Edycja danych uczestników
* Wyświetlanie danych uczestników konferencji
* Wyświetlanie danych o płatnościach
* Rezerwacja dodatkowych miejsc na konferencji
* Rezygnacja z miejsc na konferencji

## Uczestnik

* Edycja swoich danych (wybranych)
* Rejestracja na dane dni konferencji
* Rejestracja na warsztaty
* Podgląd listy warsztatów
* Podgląd listy konferencji

## Organizator

* Tworzenie konferencji
* Tworzenie warsztatów
* Edycja cennika
* Edytowanie konferencji
* Edytowanie warsztatów
* Usuwanie konferencji
* Usuwanie warsztatów
* Generowanie listy nadchodzących konferencji
* Generowanie listy nadchodzących warsztatów
* Wyświetlanie danych klienta
* Anulowanie rezerwacji na konferencje
* Generowanie listy uczestników konferencji
* Generowanie listy uczestników warsztatów
* Generowanie listy płatności
* Generowanie listy klientów
* Generowanie listy zrealizowanych konferencji / warsztatów

# Generator

Dane zapczerpnęliśmy z strony mackaroo.com generując losowe dane. Następnie wszystko zaimportowaliśmy do bazy. Ilość wierszy danej kolumny:

*client 100*

*attendee(students) 1000(250)*

*conference 50*

*workshop 200*

*workshop\_reservation 200*

*conference\_day 150*

*attendee\_lists 1000*

*reservation 500*

*price 150*

*payment 1000*

Przykładowe insert’y (całość w pliku generator.sql)

insert into attendee (attendee\_id, first\_name, last\_name) values (909, 'Ashli', 'Seniour');

insert into attendee (attendee\_id, first\_name, last\_name, student\_number) values (110, 'Ludwig', 'Lane', 109467)

insert into client (client\_id, company\_name, contact\_name, [e-mail], address, phone) values (89, 'Kain', 'Presshaugh', 'kpresshaugh2g@phoca.cz', '3 Surrey Road', '112 383 2059');

insert into conference (conference\_id, name, city, address) values (17, 'Overhold', 'Obonoma', '919 Schiller Alley');

insert into conference\_day (conference\_day\_id, conference\_id, date, max\_participants) values (42, 1, '1/8/2015', 153);

insert into price (price\_id, conference\_day\_id, value) values (104, 104, 75);

insert into workshop (workshop\_id, conference\_day\_id, name, place, max\_participants) values (139, 63, 'Center', 'Zhongzuiling', 66);

insert into reservation (reservation\_id, client\_id, conference\_day\_id, places\_reserved, reservation\_date) values (387, 35, 84, 147, '6/5/2015');

insert into workshop\_reservation (workshop\_reservation\_id, workshop\_id, reservation\_id, places\_reserved) values (103, 197, 481, 153);

insert into workshop\_attendee\_list (workshop\_attendee\_list\_id, attendee\_id, workshop\_reservation\_id) values (123, 123, 142);

insert into conference\_attendee\_list (conference\_attendee\_list\_id, reservation\_id, attendee\_id, conference\_day\_id) values (851, 63, 827, 60);

insert into payment (payment\_id, reservation\_id, payment\_date, value) values (999, 277, '1/12/2017', 889);