Projekt:

Konferencje

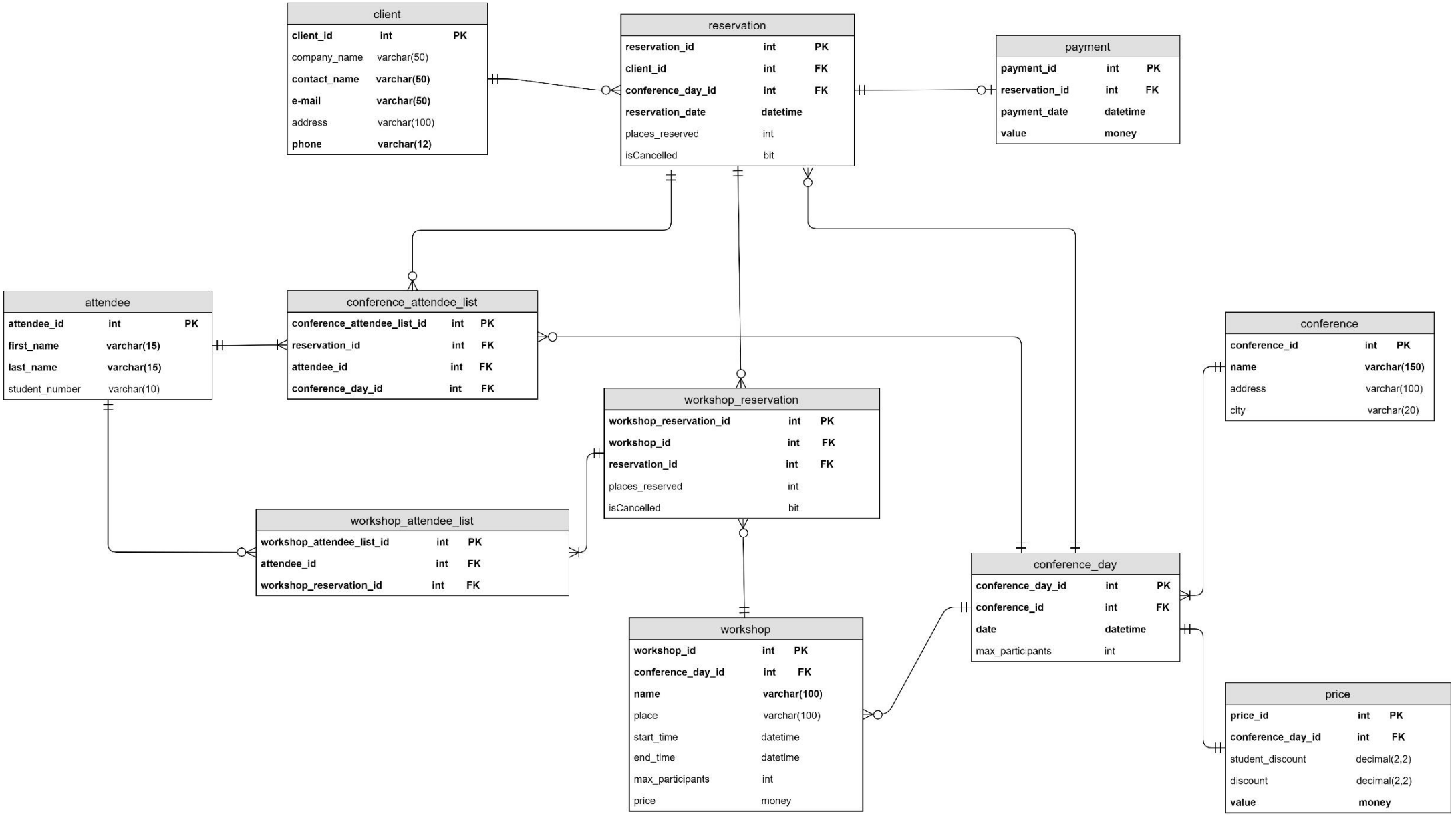
Oscar Teeninga & Kamil Szarek

Podstawy baz danych

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

Spis treści

Schemat bazy danych



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 ugłę 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
```

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);
```