# Bazy Konferencje

Krzysztof Piaskowy, Jakub Płotnikowski

2018/2019

Projekt bazy danych dla firmy organizującej konferencje

## Spis treści

**1 Wprowadzenie** **4**

**2 Tabele** **5**
2.1 Tabela Clients . . . . . 5
2.2 Tabela Company . . . . . 7
2.3 Tabela Conferences . . . . . 8
2.4 Tabela Conferences_days . . . . . 9
2.5 Tabela Conferences_days_bookings . . . . . 10
2.6 Tabela Conferences_days_registrations . . . . . 11
2.7 Tabela Participants . . . . . 12
2.8 Tabela Payments . . . . . 13
2.9 Tabela Price_levels . . . . . 14
2.10 Tabela Workshops . . . . . 15
2.11 Tabela Workshops_booking . . . . . 16
2.12 Tabela Workshops_registrations . . . . . 17

**3 Widoki** **18**
3.1 Widok attendance_card . . . . . 18
3.2 Widok canceled_conferences_booking . . . . . 19
3.3 Widok canceled_workshop_booking . . . . . 20
3.4 Widok client_paid_statistic . . . . . 21
3.5 Widok clients_full_data . . . . . 22
3.6 Widok company_clients_full_data . . . . . 23
3.7 Widok conferences_attendances . . . . . 24
3.8 Widok conferences_workshop . . . . . 25
3.9 Widok individual_clients_full_data . . . . . 26
3.10 Widok past_conferences . . . . . 27
3.11 Widok payments_saldo . . . . . 28
3.12 Widok people_who_did_not_fill_in_their_personal_Data_within_two_weeks 29
3.13 Widok present_and_future_conferences . . . . . 30
3.14 Widok top100_most_attended_conferences . . . . . 31
3.15 Widok top100_most_attended_conferences_day . . . . . 32
3.16 Widok top100_most_attended_workshops . . . . . 33
3.17 Widok workshop_attendances . . . . . 34

**4 Procedury** **35**
4.1 Procedura addClientAsCompany . . . . . 35
4.2 Procedura addClientAsIndividual . . . . . 37
4.3 Procedura addConferenceRegistrationForParticipant . . . . . 38
4.4 Procedura addNewCompany . . . . . 39
4.5 Procedura addNewConference . . . . . 40
4.6 Procedura addNewConferenceDay . . . . . 41
4.7 Procedura addNewConferenceDayBooking . . . . . 42
4.8 Procedura addNewParticipant . . . . . 43
4.9 Procedura addNewPayment . . . . . 44
4.10 Procedura addNewPriceLevel . . . . . 45
4.11 Procedura addNewWorkshop . . . . . 46
4.12 Procedura addNewWorkshopBooking . . . . . 47
4.13 Procedura addNewWorkshopRegistration . . . . . 49
4.14 Procedura addWorkshopRegistrationForParticipant . . . . . 51
4.15 Procedura cancelConferencesDayBooking . . . . . 52
4.16 Procedura cancelConferencesDayRegistration . . . . . 53
4.17 Procedura cancelWorkshopBooking . . . . . 54
4.18 Procedura cancelWorkshopRegistration . . . . . 55
4.19 Procedura updateClient . . . . . 56
4.20 Procedura updateCompany . . . . . 58
4.21 Procedura updateConference . . . . . 59
4.22 Procedura updateConferenceDay . . . . . 60
4.23 Procedura updateParticipant . . . . . 61
4.24 Procedura updatePrizeLevel . . . . . 62
4.25 Procedura updateWorkshop . . . . . 63

**5 Funkcje** **64**
5.1 Funkcja doWorkshopsOverlapEachOther . . . . . 64
5.2 Funkcja endTimeOfWorkshop . . . . . 65
5.3 Funkcja getPaidAmountByClient . . . . . 66
5.4 Funkcja getPriceOfConferencesDayByConferencesDayBookingId . . . . . 67
5.5 Funkcja getStudentDiscount . . . . . 68
5.6 Funkcja getWorkshopPriceById . . . . . 69
5.7 Funkcja listOfWorkshopsForClient . . . . . 70
5.8 Funkcja listOfWorkshopsForClient . . . . . 71

**6 Triggery** **72**
6.1 Trigger checkDataForClient . . . . . 72
6.2 Trigger checkDataForCompany . . . . . 73
6.3 Trigger blockDayThatExceedsConference . . . . . 74
6.4 Trigger checkDateForConferenceDay . . . . . 75
6.5 Trigger checkBookingDateForConferenceDay . . . . . 76
6.6 Trigger notEnoughPlacesForConferenceDay . . . . . 77
6.7 Trigger checkDataForParticipant . . . . . 78
6.8 Trigger checkDataForPayment . . . . . 79
6.9 Trigger checkPaymentDate . . . . . 80
6.10 Trigger checkDataForPrizeLevel . . . . . 81
6.11 Trigger checkDataForWorkshop . . . . . 82

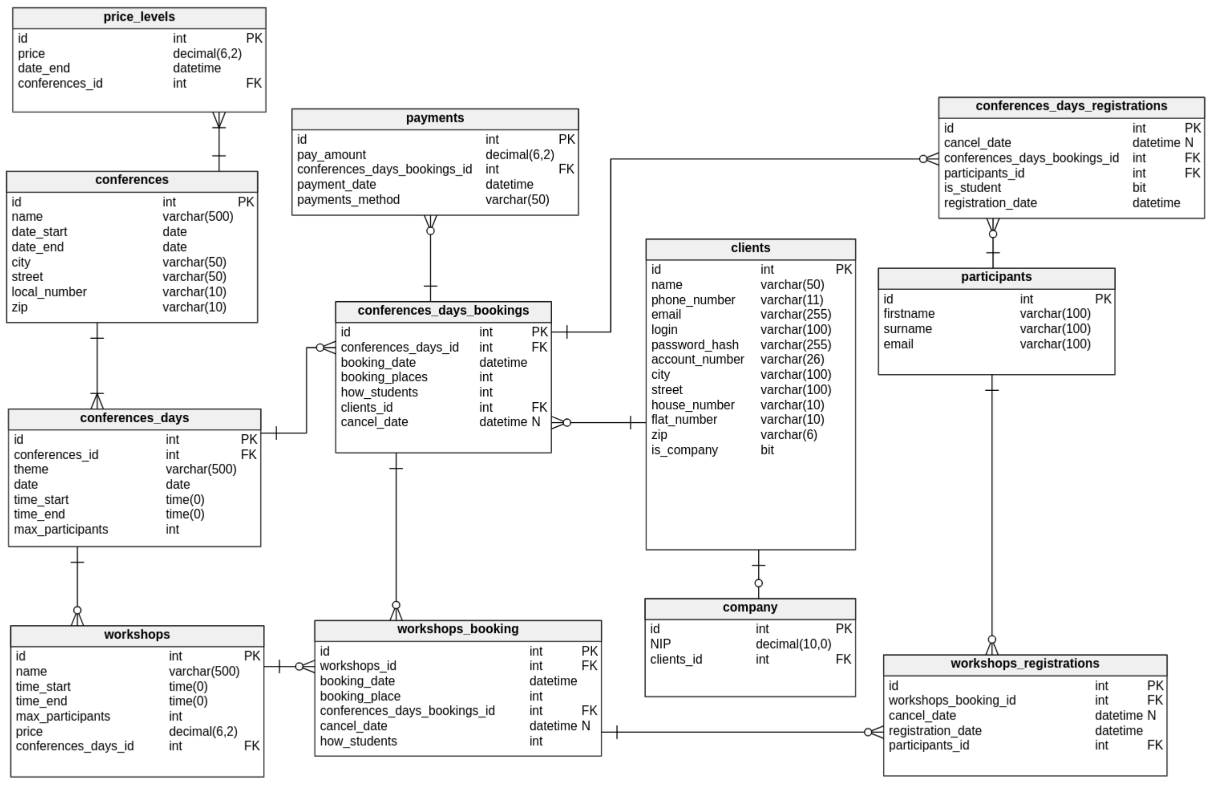
6.12 Trigger checkDataForWorkshopBooking . . . 83
6.13 Trigger morePlacesBookedForWorkshopThanForConferenceDay . . . 84
6.14 Trigger notEnoughPlacesForWorkshop . . . 85
6.15 Trigger tooManyStudents . . . 86
6.16 Trigger checkMaxBookingPlaces . . . 87
6.17 Trigger checkWhetherClientIsAlreadyBookedForAnotherWorkshop . . . 88
6.18 Trigger checkWorkshopRegistration . . . 89
6.19 Trigger participantExistsInWorkshop . . . 90

**7 Indeksy** **91**
7.1 Indeksy dla tabeli clients . . . 91
7.2 Indeksy dla tabeli company . . . 91
7.3 Indeksy dla tabeli conferences . . . 91
7.4 Indeksy dla tabeli conferences_days . . . 92
7.5 Indeksy dla tabeli conferences_days_bookings . . . 92
7.6 Indeksy dla tabeli conferences_days_registrations . . . 92
7.7 Indeksy dla tabeli participants . . . 92
7.8 Indeksy dla tabeli payments . . . 93
7.9 Indeksy dla tabeli price_levels . . . 93
7.10 Indeksy dla tabeli workshops . . . 93
7.11 Indeksy dla tabeli workshops_booking . . . 93

**8 Role** **94**
8.1 Administrator . . . 94
8.2 Pracownik firmy . . . 94
8.3 Klient . . . 94

**9 Generator danych** **95**

## 1 Wprowadzenie



Rysunek 1: Schemat bazy danych

# 2 Tabele

Wykaz tabel w bazie danych wraz z opisem ich zawartości.

## 2.1 Tabela Clients

Tabela zawierająca informacje na temat klientów którzy mogą rezerwować miejsca na konferencje.
Pola występujące w tabeli:

- id - Primary Key
- name - Nazwa klienta
- phone_number - numer kontaktowy klienta
- email - adres email klienta
- login - login klinta do konta w systemie
- password_hash - hash hasła użytkownika
- account_number - numer konta bankowego
- city - miasto w którym mieszka klient
- street - ulica na której mieszka klient
- house_number - numer domu klienta
- flat_number - numer mieszkania klienta
- zip - kod pocztowy klienta
- is_company - flaga oznaczająca czy klient jest kilnetem indywidualnym czy firmą

Listing 1: Tabela clients

```
CREATE TABLE clients (
    id int NOT NULL IDENTITY,
    name varchar(50) NOT NULL,
    phone_number varchar(11) NOT NULL,
    email varchar(255) NOT NULL,
    login varchar(100) NOT NULL,
    password_hash varchar(255) NOT NULL,
    account_number varchar(26) NOT NULL,
    city varchar(100) NOT NULL,
    street varchar(100) NOT NULL,
    house_number varchar(10) NOT NULL,
    flat_number varchar(10) NOT NULL,
    zip varchar(6) NOT NULL,
    is_company bit NOT NULL,
    CONSTRAINT clients_email_uindex UNIQUE (email),
    CONSTRAINT clients_account_number_uindex UNIQUE (account_number),
    CONSTRAINT clients_phone_number_uindex UNIQUE (phone_number),
    CONSTRAINT cl_checkCompany CHECK (is_company = 0 or is_company = 1),
    CONSTRAINT cl_checkZip CHECK (zip like '[0-9][0-9]-[0-9][0-9][0-9]'),
    CONSTRAINT cl_checkHouseNumber CHECK (house_number not like '%[^0-9]%'),
    CONSTRAINT cl_checkPhoneNumber CHECK (phone_number not like '%[^0-9]%'),
    CONSTRAINT cl_checkEmail CHECK (email like '%_@_%._%'),
    CONSTRAINT clients_pk PRIMARY KEY (id)
);
```

## 2.2 Tabela Company

Tabela zawiera informacje na temat firmy
Pola występujące w tabeli:

- id - Primary Key
- NIP - numer NIP firmy
- clients_id - id klienta

Listing 2: Tabela company

```sql
CREATE TABLE company (
    id int NOT NULL IDENTITY,
    NIP decimal(10,0) NOT NULL,
    clients_id int NOT NULL,
    CONSTRAINT company_NIP_uindex UNIQUE (NIP),
    CONSTRAINT company_clients_id_uindex UNIQUE (clients_id),
    CONSTRAINT company_pk PRIMARY KEY (id)
);
```

## 2.3 Tabela Conferences

Tabela zawiera informacje na temat konferencji
Pola występujące w tabeli:

- id - Primary Key
- name - nazwa konferencji
- date_start - data rozpoczęcia konferencji
- date_end - data zakończenia konferencji
- city - miasto w którym odbywa się konferencja
- street - ulica przy której odbywa się konferencja
- local_number - numer budynku w którym odbywa się konferencja
- zip - kod pocztowy miasta w którym odbywa się konferencja

Listing 3: Tabela conferences

```sql
CREATE TABLE conferences (
    id int NOT NULL IDENTITY,
    name varchar(500) NOT NULL,
    date_start date NOT NULL,
    date_end date NOT NULL,
    city varchar(50) NOT NULL,
    street varchar(50) NOT NULL,
    local_number varchar(10) NOT NULL,
    zip varchar(10) NOT NULL,
    CONSTRAINT conf_checkEndDateNotLessByStartData CHECK (date_end >= date_start),
    CONSTRAINT conf_checkZip CHECK (zip like '[0-9][0-9]-[0-9][0-9][0-9]'),
    CONSTRAINT conferences_pk PRIMARY KEY (id)
);
```

## 2.4 Tabela Conferences_days

Tabela zawiera informacje na temat poszczególnych dni konferencji
Pola występujące w tabeli:

- id - Primary Key
- conferences_id - id konferencji
- theme - tytuł konferencji
- date - data konferencji
- time_start - godzina rozpoczęcia konferencji w danym dniu
- time_end - godzina zakończenia konferencji w danym dniu
- max_participants - maksymalna liczba uczestników dostępna dla danego dnia konferencji

Listing 4: Tabela conferences_days

```
CREATE TABLE conferences_days (
    id int NOT NULL IDENTITY,
    conferences_id int NOT NULL,
    theme varchar(500) NOT NULL,
    date date NOT NULL,
    time_start time(0) NOT NULL,
    time_end time(0) NOT NULL,
    max_participants int NOT NULL,
    CONSTRAINT confd_checkMaxParticipants CHECK (max_participants > 0),
    CONSTRAINT confd_checkEndDateNotLessByStartData CHECK (time_end >= time_start),
    CONSTRAINT conferences_days_pk PRIMARY KEY (id)
);
```

## 2.5 Tabela Conferences_days_bookings

Tabela zawiera informacje na temat rezerwacji na poszczególne dni konferencji
Pola występujące w tabeli:

- id - Primary Key
- conferences_days_id - id konkretnego dnia konferencji
- booking_date - data dokonania rezerwacji konferencji
- booking_places - liczba zarezerwowanych miejsc
- how_students - liczba studentów w spośród zarezerwowanych
- clients_id - id klienta konferencji
- camcel_date - data rezygnacji z rezerwacji

Listing 5: Tabela conferences_days_bookings

```
CREATE TABLE conferences_days_bookings (
   id int NOT NULL IDENTITY,
   conferences_days_id int NOT NULL,
   booking_date datetime NOT NULL,
   booking_places int NOT NULL,
   how_students int NOT NULL,
   clients_id int NOT NULL,
   cancel_date datetime NULL,
   CONSTRAINT confdb_checkBookingDate CHECK (cancel_date >= booking_date ),
   CONSTRAINT confdb_checkHowManyStudents CHECK (how_students >= 0),
   CONSTRAINT confdb_checkBookingPlaces CHECK (booking_places >= 0),
   CONSTRAINT conferences_days_bookings_pk PRIMARY KEY (id)
);
```

## 2.6 Tabela Conferences_days_registrations

Tabela zawiera informacje na temat rejestracji na poszczególne dni konferencji
Pola występujące w tabeli:

- id - Primary Key
- cancel_date - godzina rezygnacji z danego dnia konferencji
- conferences_days_bookings_id - id rezerwacji na konkretny dzień konferencji
- participants_id - id uczestnika konferencji
- is_student - informacja o tym czy uczestnik jest studentem (1 - jest studentem, 0 - nie jest studentem)

Listing 6: Tabela conferences_days_registrations

```
CREATE TABLE conferences_days_registrations (
    id int NOT NULL IDENTITY,
    cancel_date datetime NULL,
    conferences_days_bookings_id int NOT NULL,
    participants_id int NOT NULL,
    is_student bit NOT NULL,
    registration_date datetime NOT NULL,
    CONSTRAINT confdr_checkStudent CHECK (is_student = 0 or is_student = 1),
    CONSTRAINT confdr_checkDates CHECK (cancel_date > registration_date ),
    CONSTRAINT conferences_days_registrations_pk PRIMARY KEY (id)
);
```

## 2.7 Tabela Participants

Tabela zawiera informacje na temat uczestników konferencji
Pola występujące w tabeli:

- id - Primary Key
- firstname - imię uczestnika konferencji
- surname - nazwisko uczestnika konferencji
- email - adres email uczestnika konferencji

Listing 7: Tabela Participants

```
CREATE TABLE participants (
    id int NOT NULL IDENTITY,
    firstname varchar(100) NOT NULL,
    surname varchar(100) NOT NULL,
    email varchar(100) NOT NULL,
    CONSTRAINT participants_pk PRIMARY KEY (id)
);
```

## 2.8 Tabela Payments

Tabela zawiera informacje na temat opłat za konferencje i warsztaty
Pola występujące w tabeli:

- id - Primary Key
- pay_amount - kwota zapłacona
- conferences_days_bookings_id - id rezerwacji na konkretny dzień konferencji
- date - data wykonania opłaty
- payments_method - sposób wykonania opłaty

Listing 8: Tabela payments

```
CREATE TABLE payments (
    id int NOT NULL IDENTITY,
    pay_amount decimal(6,2) NOT NULL,
    conferences_days_bookings_id int NOT NULL,
    payment_date datetime NOT NULL,
    payments_method varchar(50) NOT NULL,
    CONSTRAINT p_checkPayAmount CHECK (pay_amount >= 0),
    CONSTRAINT payments_pk PRIMARY KEY (id)
);
```

## 2.9 Tabela Price_levels

Tabela zawiera informacje na progów cenowych
Pola występujące w tabeli:

- id - Primary Key
- price - wartość progu cenowego
- date_start - data rozpoczęcia okresu w którym obowiązuje dany próg cenowy
- date_end - data zakończenia okresu w którym obowiązuje dany próg cenowy
- conferences_id - id konferencji

Listing 9: Tabela price_levels

```
CREATE TABLE price_levels (
    id int NOT NULL IDENTITY,
    price decimal(6,2) NOT NULL,
    date_end datetime NOT NULL,
    conferences_id int NOT NULL,
    CONSTRAINT pl_checkPrice CHECK (price >= 0),
    CONSTRAINT price_levels_pk PRIMARY KEY (id)
);
```

## 2.10 Tabela Workshops

Tabela zawiera informacje na temat warsztatów
Pola występujące w tabeli:

- id - Primary Key
- name - nazwa warsztatu
- time_start - godzina rozpoczęcia warsztatu
- time_end - godzina zakończenia warsztatu
- max_participants - maksymalna liczba uczestników jaka może uczestniczyć w danym warsztacie
- price - cena danego warsztatu
- conferences_days_id - id dnia konferencji podczas którego odbywa się dany warsztat

Listing 10: Tabela workshops

```
CREATE TABLE workshops (
    id int NOT NULL IDENTITY,
    name varchar(500) NOT NULL,
    time_start time(0) NOT NULL,
    time_end time(0) NOT NULL,
    max_participants int NOT NULL,
    price decimal(6,2) NOT NULL,
    conferences_days_id int NOT NULL,
    CONSTRAINT w_checkEndDateNotLessByStartData CHECK (time_end >= time_start),
    CONSTRAINT w_checkMaxParticipants CHECK (max_participants > 0),
    CONSTRAINT w_checkPrice CHECK (price >= 0),
    CONSTRAINT workshops_pk PRIMARY KEY (id)
);
```

## 2.11 Tabela Workshops_booking

Tabela zawiera informacje na temat rezerwacji na warsztaty
Pola występujące w tabeli:

- id - Primary Key
- workshops_id - id warsztatu na który składana jest rezerwacja
- booking_date - data wykonania rezerwacji na dany warsztat
- booking_place - ilość zarezerwowanych miejsc przez klienta na daną konferencję albo warsztat
- to_pay - kwota jaka została do zapłaty za dany warsztat
- conferences_days_bookings_id - id rezerwacji jaka została dokonana na dany dzień konferencji
- how_students - luczba studentów wśród zarezerwowanych miejsc

Listing 11: Tabela workshops_booking

```
CREATE TABLE workshops_booking (
   id int NOT NULL IDENTITY,
   workshops_id int NOT NULL,
   booking_date datetime NOT NULL,
   booking_place int NOT NULL,
   conferences_days_bookings_id int NOT NULL,
   cancel_date datetime NULL,
   how_students int NOT NULL,
   CONSTRAINT workshops_booking_pk PRIMARY KEY (id)
);
```

## 2.12 Tabela Workshops_registrations

Tabela zawiera informacje na temat rejestracji na warsztaty
Pola występujące w tabeli:

- id - Primary Key
- workshops_booking_id - id rezerwacji warsztatu
- cancel_date - data anulowania rejestracji
- conferences_days_registrations_id - id rejestracji na konkretny dzień konferencji

Listing 12: Tabela workshops_registrations

```
CREATE TABLE workshops_registrations (
    id int NOT NULL IDENTITY,
    workshops_booking_id int NOT NULL,
    cancel_date datetime NULL,
    registration_date datetime NOT NULL,
    participants_id int NOT NULL,
    CONSTRAINT workr_checkDates CHECK ( cancel_date > registration_date),
    CONSTRAINT workshops_registrations_pk PRIMARY KEY (id)
);
```

# 3 Widoki

## 3.1 Widok attendance_card

Widok generujący dane do identyfikatora uczestnika konferencji.

Listing 13: Widok attendance_card

```sql
create view attendance_card
  as
select c.name conferences_name,
       cd.theme theme_conference_day,
       cd.date conferences_day_date,
       cd.time_start conferences_time_start,
       cd.time_end conferences_time_end,
       p.firstname attendance_name,
       p.surname attendance_surname
from conferences_days_registrations cdr
join participants p on cdr.participants_id = p.id
join conferences_days_bookings cdb on cdr.conferences_days_bookings_id = cdb.id
join conferences_days cd on cdb.conferences_days_id = cd.id
join conferences c on cd.conferences_id = c.id
go
```

## 3.2 Widok canceled_conferences_booking

Widok prezentujący anulowane konferencje

Listing 14: Widok canceled_conferences_booking

```
create view canceled_conferences_booking
  as
select c.name client_name,
       c.email client_email,
       cdb.cancel_date cancel_date,
       cd.theme theme_conferences_day,
       c2.name coferences_name
from clients c
       join conferences_days_bookings cdb on c.id = cdb.clients_id
       join conferences_days cd on cdb.conferences_days_id = cd.id
       join conferences c2 on cd.conferences_id = c2.id
where cancel_date is not null
go
```

## 3.3 Widok canceled_workshop_booking

Widok prezentujący anulowane warsztaty

Listing 15: Widok canceled_workshop_booking

```
create view canceled_workshop_booking
  as
select c.name client_name,
      c.email client_email,
      wb.cancel_date cancel_date,
      w.name workshop_name
from clients c
      join conferences_days_bookings cdb on c.id = cdb.clients_id
      join workshops_booking wb on cdb.id = wb.conferences_days_bookings_id
      join workshops w on wb.workshops_id = w.id
where wb.cancel_date is not null
go
```

## 3.4 Widok client_paid_statistic

Widok prezentujący statystyki klientów odnośnie płatności

Listing 16: Widok client_paid_statistic

```sql
create view client_paid_statistic
  as
select c.name,
       c.city,
       sum(p.pay_amount) total_paid
from clients c
join conferences_days_bookings cdb on c.id = cdb.clients_id
join payments p on cdb.id = p.conferences_days_bookings_id
group by c.id, c.name, c.city
go
```

## 3.5 Widok clients_full_data

Widok prezentujący całość danych o klientach z tabeli clients

Listing 17: Widok clients_full_data

```
create view clients_full_data as
select name,
      phone_number,
      email,
      login,
      account_number,
      city,
      street,
      house_number,
      flat_number,
      zip,
      is_company
from clients
go
```

## 3.6 Widok company_clients_full_data

Widok prezentujący wszystkie dane na temat klientów, którzy są firmą

Listing 18: Widok company_clients_full_data

```
create view company_clients_full_data as
select name,
      phone_number,
      email,
      login,
      account_number,
      city,
      street,
      house_number,
      flat_number,
      zip,
      is_company,
      NIP,
      clients_id
from clients
      left join company as c on clients.id = c.clients_id
where is_company = 1
go
```

## 3.7 Widok conferences_attendances

Widok prezentujący uczestników danej konferencji oraz informacje o nich.

Listing 19: Widok conferences_attendances

```
CREATE view conferences_attendances
as
select c.id         conferences_id,
       cd.id        conferences_day_id,
       cdb.clients_id clients_id,
       c.name       conferences_name,
       cd.theme     conferences_day_name,
       cd.date      conferences_date,
       p.firstname  attendent_firstname,
       p.surname    attendent_surname
from conferences c
       join conferences_days cd on c.id = cd.conferences_id
       join conferences_days_bookings cdb on cd.id = cdb.conferences_days_id and
           cdb.cancel_date is null
       join conferences_days_registrations cdr on cdb.id =
           cdr.conferences_days_bookings_id and cdr.cancel_date is null
       join participants p on cdr.participants_id = p.id
go
```

## 3.8 Widok conferences_workshop

Widok prezentujący warsztaty przypisane do danej konferencji.

Listing 20: Widok conferences_workshop

```
create view conferences_workshop
as
select c.id    conferences_id,
       cd.id   conferences_day_id,
       w.id    workshop_id,
       c.name  conferences_name,
       cd.theme conferences_day_theme,
       w.name  workshop_name
from conferences c
       join conferences_days cd on c.id = cd.conferences_id
       join workshops w on cd.id = w.conferences_days_id
go
```

## 3.9 Widok individual_clients_full_data

Widok prezentujący dane na temat indywidualnych klientów

Listing 21: Widok individual_clients_full_data

```
create view individual_clients_full_data as
 select name,
       phone_number,
       email,
       login,
       account_number,
       city,
       street,
       house_number,
       flat_number,
       zip,
       is_company
from clients
where is_company = 0
go
```

## 3.10 Widok past_conferences

Widok prezentujący listę konferencji, które się już odbyły

Listing 22: Widok past_conferences

```
CREATE view past_conferences as
select name, date_start, date_end, city, street, local_number, zip
from conferences
where conferences.date_end < getdate()
go
```

## 3.11 Widok payments_saldo

Widok prezentujący saldo płatności dla danego klienta

Listing 23: Widok payments_saldo

```
CREATE view payments_saldo
  as
select c.name,
       c.id,
       sum(isnull(dbo.getWorkshopPriceById(wb.workshops_id), 0) * wb.booking_place -
           dbo.getStudentDiscount() * wb.how_students *
               isnull(dbo.getWorkshopPriceById(wb.workshops_id), 0)
             +
           isnull(dbo.getPriceOfConferencesDayByConferencesDayBookingId(cdb.id), 0) -
           dbo.getStudentDiscount() * cdb.how_students *
               isnull(dbo.getPriceOfConferencesDayByConferencesDayBookingId(cdb.id),
               0))
             -
           isnull(dbo.getPaidAmountByClient(c.id), 0) paymentsSaldo
from clients c
       join conferences_days_bookings cdb on c.id = cdb.clients_id
       join workshops_booking wb on cdb.id = wb.conferences_days_bookings_id
group by c.id, c.name
go
```

## 3.12 Widok people_who_did_not_fill_in_their_personal_Data_within_two_weeks

Widok prezentujący klientów którzy nie podali danych osobowych po 2 tygodniach od rezerwacji.

Listing 24: Widok people_who_did_not_fill_in_their_personal_Data_within_two_weeks

```
CREATE view people_who_did_not_fill_in_their_personal_Data_within_two_weeks as
   select c.name,
         c.phone_number,
         c.email,
         c.login,
         c.account_number,
         c.city,
         c.street,
         c.house_number,
         c.flat_number,
         c.zip
   from clients c
         join conferences_days_bookings b on c.id = b.clients_id
         join workshops_booking wb on b.id = wb.conferences_days_bookings_id
   where wb.booking_place >
        isnull((select count(*)
               from workshops_registrations wr2
               where wr2.workshops_booking_id = wb.id
               group by wr2.workshops_booking_id), 0)
go
```

### 3.13 Widok present_and_future_conferences

Widok prezentujący listę trwających i przyszłych konferencji

Listing 25: Widok present_and_future_conferences

```
create view present_and_future_conferences as
select c.name,
      c.date_start,
      c.date_end,
      c.city,
      c.street,
      c.local_number,
      c.zip
from conferences as c
where c.date_start >= getdate()
group by c.name, date_start, date_end, city, street, local_number, zip
go
```

## 3.14 Widok top100_most_attended_conferences

Widok prezentujący 100 najchętniej odwiedzanych konferencji

Listing 26: Widok top100_most_attended_conferences

```
CREATE view top100_most_attended_conferences as
  select top 100 name, date_start, date_end, city, street, local_number, zip,
      cdb.booking_places
  from conferences
        left join conferences_days on conferences.id = conferences_days.conferences_id
        left join conferences_days_bookings cdb on conferences_days.id =
            cdb.conferences_days_id
  group by name, date_start, date_end, city, street, local_number, zip,
      cdb.booking_places
  order by booking_places desc
go
```

## 3.15 Widok top100_most_attended_conferences_day

Widok prezentujący 100 najchętniej odwiedzanych dni konferencji

Listing 27: Widok top100_most_attended_conferences_day

```
create view top100_most_attended_conferences_days as
select top 100 theme, date, time_start, time_end, max_participants, cdb.booking_places
from conferences_days
      left join conferences_days_bookings cdb on conferences_days.id =
          cdb.conferences_days_id
order by booking_places desc
go
```

## 3.16 Widok top100_most_attended_workshops

Widok prezentujący 100 najchętniej odwiedzanych warsztatów

Listing 28: Widok top100_most_attended_workshops

```sql
CREATE view top100_most_attended_workshops as
select top 100 name, time_start, time_end, max_participants, price, wb.booking_place
from workshops
      left join workshops_booking wb on workshops.id = wb.workshops_id
order by booking_place desc
go
```

## 3.17 Widok workshop_attendances

Widok prezentujący oczestników danego warsztatu

Listing 29: Widok workshop_attendances

```
CREATE view workshop_attendances
as
select c.id      conferences_id,
      cd.id      conferences_day_id,
      w.id       workshop_ic,
      c.name     conferences_name,
      cd.theme   conferences_day_theme,
      p.firstname firstname,
      p.surname  surname,
      p.email    email
from conferences c
      join conferences_days cd on c.id = cd.conferences_id
      join workshops w on cd.id = w.conferences_days_id
      join workshops_booking wb on w.id = wb.workshops_id and wb.cancel_date is null
      join workshops_registrations wr on wb.id = wr.workshops_booking_id and
          wr.cancel_date is null
      join participants p on wr.participants_id = p.id
group by c.id, c.name, cd.id, theme, w.id, p.id, firstname, surname, email
go
```

## 4 Procedury

Wykaz procedur, które można wykorzystywać korzystając z bazy danych

### 4.1 Procedura addClientAsCompany

Procedura pozwalająca dodac nowego klienta (firme) do bazy danych

Listing 30: Procedura addClientAsCompany

```
CREATE procedure addClientAsCompany @name varchar(50), @phone_number varchar(11),
    @email varchar(255),
                                  @login varchar(100), @password_hash varchar(255),
                                  @account_number varchar(26), @city varchar(100),
                                  @street varchar(100), @house_number varchar(10),
                                  @flat_number varchar(10), @zip varchar(6),
                                  @NIP decimal(10, 0)
as
begin
  set nocount on;
  begin try
    if exists(
        select *
        from clients
        where login = @login
      )
      begin
        throw 50001, 'Login already exists.', 1;
      end
    if exists(
        select *
        from clients
        where email = @email
      )
      begin
        throw 50002, 'Mail is already used.', 2;
      end
    insert into clients (name, phone_number, email, login, password_hash,
        account_number, city, street, house_number,
                       flat_number, zip, is_company)
    values (@name, @phone_number, @email, @login, @password_hash, @account_number,
        @city, @street, @house_number,
           @flat_number, @zip, 1)
    if not exists(
        select * from company where NIP = @NIP
      )
      begin
        insert into company
          (NIP, clients_id)
        values (@NIP, (select id from clients where email = @email))
      end
  end try
  begin catch
    declare @errorMessage nvarchar(2048)
      = 'Cannot add company client, error: ' + error_message();
    throw 50003, @errorMessage, 3;
  end catch
end
go
```

## 4.2 Procedura addClientAsIndividual

Procedura pozwalająca dodac nowego klienta (indywidualnego) do bazy danych

Listing 31: Procedura addClientAsIndividual

```
create procedure addClientAsIndividual @name varchar(50), @phone_number varchar(11),
    @email varchar(255),
                                   @login varchar(100), @password_hash varchar(255),
                                   @account_number varchar(26), @city varchar(100),
                                   @street varchar(100), @house_number varchar(10),
                                   @flat_number varchar(10), @zip varchar(6)

as
begin
  set nocount on;
  begin try
    if exists(
        select *
        from clients
        where login = @login
      )
      begin
        throw 50001, 'Login already exists.', 1;
      end
    if exists(
        select *
        from clients
        where email = @email
      )
      begin
        throw 50002, 'Mail is already used.', 2;
      end
    insert into clients (name, phone_number, email, login, password_hash,
        account_number, city, street, house_number,
                        flat_number, zip, is_company)
    values (@name, @phone_number, @email, @login, @password_hash, @account_number,
        @city, @street, @house_number,
            @flat_number, @zip, 0)
  end try
  begin catch
    declare @errorMessage nvarchar(2048)
      = 'Cannot add individual client, error: ' + error_message();
    throw 50003, @errorMessage, 3;
  end catch
end
go
```

## 4.3 Procedura addConferenceRegistrationForParticipant

Procedura pozwalająca dodać rejestrację konferencji dla uczestnika

Listing 32: Procedura addConferenceRegistrationForParticipant

```
CREATE procedure addConferenceRegistrationForParticipant @firstname varchar(100),
    @surname varchar(100),
  @email varchar(100),
  @conferences_days_booking_id int, @is_student bit
  as
  begin
    set nocount on
    begin try
      if not exists
        (
          select *
          from participants
          where firstname = @firstname
            and surname = @surname
            and email = @email
        )
        insert into participants (firstname, surname, email)
        values (@firstname, @surname, @email)
      if not exists
        (
          select *
          from conferences_days_bookings cdb
          where cdb.id = @conferences_days_booking_id
        )
        begin
          ; throw 70080, 'There is no such conferences days booking id in the
              database', 1
        end
      insert into conferences_days_registrations (cancel_date,
          conferences_days_bookings_id, participants_id,
                                              is_student,
                                              registration_date)
      values (null, @conferences_days_booking_id, (select id
                                               from participants
                                               where firstname = @firstname
                                                 and surname = @surname
                                                 and email = @email), @is_student,
                                                     getdate())
    end try
    begin catch
      declare @errorMessage nvarchar(1024)
        = 'Cannot add registration for participant ' + error_message();
      throw 70090, @errorMessage, 2;
    end catch
  end
go
```

## 4.4 Procedura addNewCompany

Procedura pozwalająca dodac nową firmę do bazy danych

Listing 33: Procedura addNewCompany

```
CREATE procedure addNewCompany @NIP decimal(10, 0),
                            @clients_id int
as
begin
  set nocount on;
  begin try
    if not exists(
        select *
        from clients
        where id = @clients_id
      )
      begin
        throw 50001, 'Client does not exist.', 1;
      end
    if exists(
        select *
        from company
        where NIP = @NIP
      )
      begin
        throw 50001, 'Company already exists', 1;
      end
    insert into company
    (NIP, clients_id)
    values (@NIP, @clients_id)
  end try
  begin catch
    declare @errorMessage nvarchar(2048)
      = 'Cannot add company, error: ' + error_message();
    throw 50003, @errorMessage, 3;
  end catch
end
go
```

## 4.5 Procedura addNewConference

Procedura pozwalająca dodac nową konferencję do bazy danych

Listing 34: Procedura addNewConference

```
CREATE procedure addNewConference @name varchar(500), @date_start date,
                                   @date_end date, @city varchar(50),
                                   @street varchar(50), @local_number varchar(10),
                                   @zip varchar(10)
  as
  begin
    set nocount on
    begin try
      insert into conferences (name, date_start, date_end, city, street,
          local_number, zip)
      values (@name, @date_start, @date_end, @city, @street, @local_number, @zip)
    end try
    begin catch
      declare @errorMessage nvarchar(1024) = 'Could not add new conference to
          database, error message' +
                                                ERROR_MESSAGE();
      ;throw 70000, @errorMessage, 1
    end catch
  end
go
```

## 4.6 Procedura addNewConferenceDay

Procedura pozwalająca dodac nowy dzień do danej konferencji w bazie danych

Listing 35: Procedura addNewConferenceDay

```
CREATE procedure addNewConferenceDay @conferences_id int,
                                  @theme varchar(500),
                                  @date date,
                                  @time_start time(0),
                                  @time_end time(0),
                                  @max_participants int
as
begin
  set nocount on
  begin try
    if not exists
      (
        select *
        from conferences
        where conferences.id = @conferences_id
      )
      begin
        ; throw 70000, 'There is no such conference in the database', 1
      end
    if exists
      (
        select *
        from conferences_days
        where conferences_days.date = @date
          and conferences_days.conferences_id = @conferences_id
      )
      begin
        ; throw 70000, 'There is already conference day with the same date', 1
      end
    if not exists
      (
        select *
        from conferences conf
        where conf.date_start <= @date and conf.date_end >= @date
      )
      begin
        ; throw 70000, 'Date is not in valid period of time for the conference', 1
      end
    insert into conferences_days (conferences_id, theme, date, time_start, time_end,
        max_participants)
    values (@conferences_id, @theme, @date, @time_start, @time_end, @max_participants)
  end try
  begin catch
    declare @errorMessage nvarchar(1024) = 'Cannot add new conference day ' +
        ERROR_MESSAGE();
        ; throw 70000, @errorMessage, 1
  end catch
end
go
```

## 4.7 Procedura addNewConferenceDayBooking

Procedura pozwalająca dodać rezerwację na dany dzień danej konferencji w bazie danych

Listing 36: Procedura addNewConferenceDayBooking

```
CREATE procedure addNewConferenceDaysBooking @conferences_days_id int, @booking_date
    date, @booking_places int,
  @how_students int, @clients_id int, @cancel_date date = null
  as
  begin
    set nocount on
    declare @conferenceStartDate date = (
      select date
      from conferences_days
      where conferences_days.id = @conferences_days_id
    )
    begin try
      if not exists
        (
          select *
          from conferences_days
                left join conferences on conferences.id = conferences_days.id
          where conferences_days.id = @conferences_days_id
        )
        begin
          ; throw 70000, 'Conference does not exist', 1
        end
      if not exists
        (
          select *
          from clients
          where clients.id = @clients_id
        )
        begin
          ; throw 70000, 'There is no such client', 1
        end
      if @conferenceStartDate < (dateadd(day, 14, @booking_date))
        begin
          throw 70000 , 'Conference cannot be booked 14 days before its start', 1
        end
      insert into conferences_days_bookings (conferences_days_id, booking_date,
          booking_places, how_students,
                                          clients_id,
                                          cancel_date)
      values (@conferences_days_id, convert(datetime, @booking_date),
          @booking_places, @how_students, @clients_id, convert(datetime, @cancel_date))
    end try
    begin catch
      declare @errorMessage nvarchar(1024) = 'Cannot add client, error: ' +
          error_message();
      ; throw 70000, @errorMessage, 1
    end catch
  end
go
```

## 4.8 Procedura addNewParticipant

Procedura pozwalająca na dodanie nowego uczestnika konferencji do bazy danych

Listing 37: Procedura addNewParticipant

```sql
create procedure addNewParticipant @firstname varchar(100), @surname varchar(100),
    @email varchar(100)
as
begin
  set nocount on
  begin try
    if exists
      (
        select * from participants where email = @email
      )
    begin
      ; throw 70007, 'Email already exists in participants table', 1
    end
    insert into participants (firstname, surname, email)
    values (@firstname, @surname, @email)
  end try
  begin catch
    declare @errorMessage nvarchar(1024)
      = 'Participant could not be added, error message: ' + error_message();
    throw 50003, @errorMessage, 1;
  end catch
end
go
```

## 4.9 Procedura addNewPayment

Procedura pozwalająca na dodanie nowej opłaty do bazy danych

Listing 38: Procedura addNewPayment

```
CREATE procedure addNewPayment @pay_amount decimal(6, 2),
    @conferences_days_booking_id int, @payment_date datetime,
                                @payments_method varchar(50)
as
begin
  set nocount on
  begin try
    if not exists
      (
        select *
        from conferences_days_bookings
        where conferences_days_bookings.id = @conferences_days_booking_id
      )
      begin
        ;throw 70010, 'Such conference day booking does not exist ', 1
      end
    if @payment_date >=
       (
        select cancel_date
        from conferences_days_bookings
        where conferences_days_bookings.id = @conferences_days_booking_id
       )
      begin
        ;throw 70011, 'Payment cannot be added to cancelled booking ', 1
      end
    insert into payments (pay_amount, conferences_days_bookings_id, payment_date,
        payments_method)
    values (@pay_amount, @conferences_days_booking_id, @payment_date,
        @payments_method)
  end try
  begin catch
    declare @errorMessage nvarchar(1024) = 'Payment could not be added, error
        message: ' + error_message()
    ;throw 70012, @errorMessage, 1
  end catch
end
go
```

## 4.10 Procedura addNewPriceLevel

Procedura pozwalająca na dodanie nowego progu cenowego do bazy danych

Listing 39: Procedura addNewPrizeLevel

```
CREATE procedure addNewPriceLevel @price decimal(6, 2),
                                  @date_end datetime,
                                  @conferences_id int
as
begin
  set nocount on;
  begin try
    if not exists
      (
        select *
        from conferences
        where id = @conferences_id
      )
      begin
        throw 50001, 'Conference does not exist', 1;
      end
    if exists
      (
        select *
        from price_levels
        where date_end = @date_end
        and id != (select id from price_levels where date_end = @date_end and
            conferences_id = @conferences_id)
      and conferences_id = @conferences_id
      )
      begin
        throw 50002, 'Busy time period', 2;
      end
    insert into price_levels (price, date_end, conferences_id)
    values (@price, @date_end, @conferences_id)
  end try
  begin catch
    declare @errorMessage nvarchar(1024) = 'Cannot add new price level, error: ' +
        ERROR_MESSAGE();
    throw 50003, @errorMessage, 3;
  end catch
end
go
```

## 4.11 Procedura addNewWorkshop

Procedura pozwalająca na dodanie warsztatu do bazy danych

Listing 40: Procedura addNewWorkshop

```
create procedure addNewWorkshop @name varchar(500), @time_start datetime, @time_end
    datetime, @max_participants int,
                                  @price decimal(6, 2), @conferences_days_id int
    as
    begin
      set nocount on
      begin try
        if not exists
          (
            select *
            from conferences_days
            where conferences_days.id = @conferences_days_id
          )
          begin
            ; throw 70000, 'There is no such conference day', 1
          end
        insert into workshops (name, time_start, time_end, max_participants, price,
            conferences_days_id)
        values (@name, @time_start, @time_end, @max_participants, @price,
            @conferences_days_id);
      end try
      begin catch
        declare @errorMessage nvarchar(1024) = 'Cannot add new workshop ' +
            ERROR_MESSAGE();
        ; throw 70000, @errorMessage, 1
      end catch
    end
go
```

## 4.12 Procedura addNewWorkshopBooking

Procedura pozwalająca na dodanie rezerwacji warsztatu do bazy danych

Listing 41: Procedura addNewWorkshopBooking

```
CREATE procedure addNewWorkshopBooking @workshop_id int,
                                   @booking_place int,
                                   @booking_date date,
                                   @conferences_day_booking_id int,
                                   @how_students int
as
begin
  set nocount on;
  begin try
    if not exists
      (
        select *
        from workshops
        where id = @workshop_id
      )
      begin
        throw 50001, 'Workshop does not exist', 1;
      end
    if not exists(
        select *
        from conferences_days_bookings
        where id = @conferences_day_booking_id
      )
      begin
        throw 50001, 'Conference day booking does not exist', 1;
      end
    if (
        (select top 1 cd.id
         from workshops w
                join conferences_days cd on w.conferences_days_id = cd.id
         where w.id = @workshop_id)
        !=
        (select top 1 cd.id
         from conferences_days_bookings cdb
                join conferences_days cd on cdb.conferences_days_id = cd.id
         where cdb.id = @conferences_day_booking_id)
      )
      begin
        throw 50001, 'Try booking workshop for another conference day.', 1;
      end
    if (@booking_place + (
      select sum(booking_place)
      from workshops_booking
      where workshops_id = @workshop_id
    )) > (select max_participants
         from workshops
         where id = @workshop_id)
      begin
        throw 50001, 'Place limit', 1
      end
    if @booking_place < @how_students
      begin
        throw 50001, 'Number of students cannot be greater than booking places', 1;
      end
    insert into workshops_booking (workshops_id, booking_date, booking_place,
        conferences_days_bookings_id, cancel_date,
                                    how_students)
    values (@workshop_id, @booking_date, @booking_place, @conferences_day_booking_id,
        NULL, @how_students)
  end try
  begin catch
    declare @errorMessage nvarchar(2048)
      = 'Cannot add workshop, error: ' + error_message();
    throw 50001, @errorMessage, 1;
  end catch
end
go
```

## 4.13 Procedura addNewWorkshopRegistration

Procedura pozwalająca na dodanie rejestracji warsztatu do bazy danych

Listing 42: Procedura addNewWorkshopRegistration

```
CREATE procedure addNewWorkshopRegistration @workshop_booking_id int,
                                            @conferences_days_registrations_id int,
                                            @registration_date datetime
as
begin
  set nocount on
  begin try
    if not exists
      (
        select *
        from conferences_days_registrations
        where id = @conferences_days_registrations_id
      )
      begin
        throw 50000, 'Conference day booking registration does not exist', 1;
      end
    if not exists
      (
        select *
        from workshops_booking
        where id = @workshop_booking_id
      )
      begin
        throw 50000, 'Workshop booking not exist', 1;
      end
    if (
        (
          select top 1 cdb.id
          from conferences_days_registrations cdr
                 join conferences_days_bookings cdb on cdr.conferences_days_bookings_id
                     = cdb.id
          where cdr.id = @conferences_days_registrations_id
        ) != (
          select top 1 c.id
          from workshops_booking wb
                 join conferences_days_bookings c on wb.conferences_days_bookings_id =
                     c.id
          where wb.id = @workshop_booking_id
        )
      )
      begin
        throw 50000, 'Trying register workshop to other conferences day', 1;
      end
    if (
           (select count(*)
            from workshops_registrations
            where workshops_booking_id = @workshop_booking_id
           ) + 1) > (select booking_place
                     from workshops_booking
                     where id = @workshop_booking_id)
      begin
        throw 70000, 'Place limit', 1
      end
    insert into workshops_registrations (workshops_booking_id,
                                         cancel_date,
                                         registration_date)
    values (@workshop_booking_id,
            NULL,
            @registration_date)
  end try
  begin catch
    declare @errorMessage nvarchar(1024) = 'Cannot add registration to conferences
        day, error: ' + error_message();
    ; throw 50000, @errorMessage, 1
  end catch
end
go
```

## 4.14 Procedura addWorkshopRegistrationForParticipant

Procedura pozwalająca na dodanie rejestracji warsztatu do bazy danych dla konkretnego uczestnika

Listing 43: Procedura addWorkshopRegistrationForParticipant

```
CREATE procedure addWorkshopRegistrationForParticipant @firstname varchar(100),
    @surname varchar(100),
                                                     @email varchar(100),
                                                     @workshops_booking_id int
as
begin
  set nocount on
  begin try
    if not exists
      (
        select *
        from participants
        where firstname = @firstname
          and surname = @surname
          and email = @email
      )
      insert into participants (firstname, surname, email)
      values (@firstname, @surname, @email)
    if not exists
      (
        select *
        from workshops_booking wb
               left join conferences_days_bookings cdb on
                   wb.conferences_days_bookings_id = cdb.id
               left join conferences_days_registrations cdr on cdb.id =
                   cdr.conferences_days_bookings_id
               left join participants p on cdr.participants_id = p.id
        where wb.id = @workshops_booking_id
      )
      begin
        ; throw 70080, 'There is no such workshops booking id in the database', 1
      end
    insert into workshops_registrations (workshops_booking_id, cancel_date,
                                         registration_date, participants_id)
    values (@workshops_booking_id, null, getdate(), (select id
                                                   from participants
                                                   where firstname = @firstname
                                                     and surname = @surname
                                                     and email = @email))
  end try
  begin catch
    declare @errorMessage nvarchar(1024)
      = 'Cannot add registration for participant ' + error_message();
    throw 70090, @errorMessage, 2;
  end catch
end
go
```

## 4.15 Procedura cancelConferencesDayBooking

Procedura pozwalająca na anulowanie rezerwacji na konkretny dzień konferencji

Listing 44: Procedura cancelConferencesDayBooking

```
CREATE procedure cancelConferencesDayBooking @id int
as
begin
  set nocount on;
  begin try
    if not exists(
        select *
        from conferences_days_bookings
        where id = @id
      )
      begin
        throw 50001, 'Conferences day not exists.', 1;
      end
    update workshops_booking
    set cancel_date = getdate()
    where id = @id
  end try
  begin catch
    declare @errorMessage nvarchar(2048)
      = 'Cannot cancel conferences day booking, error: ' + error_message();
    throw 50003, @errorMessage, 3;
  end catch
end
go
```

## 4.16 Procedura cancelConferencesDayRegistration

Procedura pozwalająca na anulowanie rejestracji na konkretny dzień konferencji

Listing 45: Procedura cancelConferencesDayRegistration

```
CREATE procedure cancelConferencesDayRegistration @id int
as
begin
  set nocount on;
  begin try
    if not exists(
        select *
        from conferences_days_registrations
        where id = @id
      )
      begin
        throw 50001, 'Conferences day registration not exists.', 1;
      end
    update conferences_days_registrations
    set cancel_date = getdate()
    where id = @id
  end try
  begin catch
    declare @errorMessage nvarchar(2048)
      = 'Cannot cancel conferences day registration, error: ' + error_message();
    throw 50003, @errorMessage, 3;
  end catch
end
go
```

## 4.17 Procedura cancelWorkshopBooking

Procedura pozwalająca na anulowanie rezerwacji na konkretny warsztat

Listing 46: Procedura cancelWorkshopBooking

```
CREATE procedure cancelWorkshopBooking @workshop_booking_id int
as
begin
  set nocount on;
  begin try
    if not exists(
        select *
        from workshops_booking
        where id = @workshop_booking_id
      )
      begin
        throw 50001, 'Workshop booking not exists.', 1;
      end
    update workshops_booking
    set cancel_date = getdate()
    where id = @workshop_booking_id
  end try
  begin catch
    declare @errorMessage nvarchar(2048)
      = 'Cannot cancel workshop booking, error: ' + error_message();
    throw 50003, @errorMessage, 3;
  end catch
end
go
```

## 4.18 Procedura cancelWorkshopRegistration

Procedura pozwalająca na anulowanie rejestracji na konkretny warsztat

Listing 47: Procedura cancelWorkshopRegistration

```
CREATE procedure cancelWorkshopRegistration @id int
as
begin
  set nocount on;
  begin try
    if not exists(
        select *
        from workshops_registrations
        where id = @id
      )
      begin
        throw 50001, 'Workshop registration not exists.', 1;
      end
    update conferences_days_registrations
    set cancel_date = getdate()
    where id = @id
  end try
  begin catch
    declare @errorMessage nvarchar(2048)
      = 'Cannot cancel workshop registration, error: ' + error_message();
    throw 50001, @errorMessage, 1;
  end catch
end
go
```

## 4.19 Procedura updateClient

Procedura służąca do aktualizowania informacji o kliencie istniejącym już w bazie danych

Listing 48: Procedura updateClient

```
create procedure updateClient @id int,
                             @name varchar(50),
                             @phone_number varchar(11),
                             @email varchar(255),
                             @login varchar(100),
                             @password_hash varchar(255),
                             @account_number varchar(26),
                             @city varchar(100),
                             @street varchar(100),
                             @house_number varchar(10),
                             @flat_number varchar(10),
                             @zip varchar(6),
                             @is_company bit
as
begin
  set nocount on;
  begin try
    if not exists
      (
        select *
        from clients
        where id = @id
      )
      begin
        throw 50001, 'There is no such client in the database', 1;
      end
    if exists(
        select *
        from clients
        where login = @login
          and @login <> (select login from clients where id = @id)
      )
      begin
        throw 50001, 'Login already exists.', 1;
      end
    if exists(
        select *
        from clients
        where email = @email
          and @email <> (select email from clients where id = @id)
      )
      begin
        throw 50001, 'Mail is already used.', 1;
      end
    update clients
    set name          = @name,
        phone_number  = @phone_number,
        email         = @email,
        login         = @login,
        password_hash = @password_hash,
        account_number = @account_number,
        city           = @city,
        street         = @street,
        house_number   = @house_number,
        flat_number    = @flat_number,
        zip            = @zip,
        is_company     = @is_company
    where id = @id
  end try
  begin catch
    declare @errorMessage nvarchar(2048) = 'Cannot update client, error: ' +
        error_message();
    throw 50001, @errorMessage, 1;
  end catch
end
go
```

## 4.20 Procedura updateCompany

Procedura służąca do aktualizowania informacji o danej firmie

Listing 49: Procedura updateCompany

```
create procedure updateCompany @id int,
                              @NIP decimal(10, 0)
as
begin
  set nocount on;
  begin try
    if exists(
        select *
        from company
        where NIP = @NIP and @NIP != (
          select NIP from company where id = @id
          )
      )
      begin
        throw 50001, 'Company already exists', 1;
      end
    update company
      set NIP          = @NIP
      where id = @id
  end try
  begin catch
    declare @errorMessage nvarchar(2048)
      = 'Cannot add company, error: ' + error_message();
    throw 50001, @errorMessage, 1;
  end catch
end
go
```

## 4.21 Procedura updateConference

Procedura służąca do aktualizowania informacji o danej konferencji

Listing 50: Procedura updateConference

```
create procedure updateConference @id int,
                               @name varchar(500),
                               @date_start datetime,
                               @date_end datetime,
                               @city varchar(50),
                               @street varchar(50),
                               @local_number varchar(10),
                               @zip varchar(10)
as
begin
  set nocount on
  begin try
    if not exists
      (
        select *
        from conferences
        where id = @id
      )
      begin
        throw 70000, 'There is no such conference in the database', 1;
      end
    update conferences
    set name = @name,
        date_start = @date_start,
        date_end = @date_end,
        city = @city,
        street = @street,
        local_number = @local_number,
        zip = @zip
    where id = @id
  end try
  begin catch
    declare @errorMessage nvarchar(1024) = 'Could not update conference to database,
        error message' + ERROR_MESSAGE();
    throw 70000, @errorMessage, 1;
  end catch
end
go
```

## 4.22 Procedura updateConferenceDay

Procedura służąca do aktualizowania informacji o danego dnia danej konferencji

Listing 51: Procedura updateConferenceDay

```
CREATE procedure updateConferenceDay @id int,
  @conferences_id int,
  @theme varchar(500),
  @date datetime,
  @time_start time(0),
  @time_end time(0),
  @max_participants int
  as
  begin
    set nocount on
    begin try
      if not exists
        (
          select *
          from conferences_days
          where id = @id
        )
        begin
          throw 70000, 'There is no such conference day in the database', 1;
        end
      if not exists
        (
          select *
          from conferences
          where conferences.id = @conferences_id
        )
        begin
          throw 70000, 'There is no such conference in the database', 1;
        end
      update conferences_days
      set theme            = @theme,
          date             = @date,
          time_start       = @time_start,
          time_end         = @time_end,
          max_participants = @max_participants
      where id = @id;
    end try
    begin catch
      declare @errorMessage nvarchar(1024) = 'Cannot update conference day ' +
          ERROR_MESSAGE();
      throw 70000, @errorMessage, 1;
    end catch
  end
go
```

## 4.23 Procedura updateParticipant

Procedura służąca do aktualizowania informacji na temat uczestnika konferencji

Listing 52: Procedura updateParticipant

```
create procedure updateParticipant @id int,
                                   @firstname varchar(100),
                                   @surname varchar(100),
                                   @email varchar(100)
as
begin
  set nocount on
  begin try
    if not exists
      (
        select * from participants where id = @id
      )
      begin
        throw 70007, 'Participant not exists in participants table', 1;
      end
    if exists
      (
        select * from participants where email = @email
      )
      begin
        throw 70007, 'Email already exists in participants table', 1;
      end
    insert into participants (firstname, surname, email)
    values (@firstname, @surname, @email)
  end try
  begin catch
    declare @errorMessage nvarchar(1024) = 'Participant could not be update, error
        message: ' + error_message();
    throw 50003, @errorMessage, 1;
  end catch
end
go
```

## 4.24 Procedura updatePrizeLevel

Procedura służąca do aktualizowania informacji na temat progu cenowego

Listing 53: Procedura updatePrizeLevel

```
create procedure updatePriceLevel @id int,
                                  @conferences_id int,
                                  @price decimal(6, 2),
                                  @date_end datetime
as
begin
  set nocount on;
  begin try
    if not exists(
        select *
        from price_levels
        where id = @id
      )
      begin
        throw 50001, 'Price levels not exist', 1;
      end
    if exists(
        select *
        from price_levels
        where date_end = @date_end
          and @date_end != (
          select date_end
          from price_levels
          where id = @id
        )
          and conferences_id = @conferences_id
      )
      begin
        throw 50002, 'Busy time period', 2;
      end
    update price_levels
    set price    = @price,
        date_end = @date_end
    where id = @id
  end try
  begin catch
    declare @errorMessage nvarchar(1024) = 'Cannot update price level, error: ' +
        ERROR_MESSAGE();
    throw 50003, @errorMessage, 3;
  end catch
end
go
```

## 4.25 Procedura updateWorkshop

Procedura służąca do aktualizowania informacji na temat warsztatu

Listing 54: Procedura updateWorkshop

```
CREATE procedure updateWorkshop @id int,
  @name varchar(500),
  @time_start datetime,
  @time_end datetime,
  @max_participants int,
  @price decimal(6, 2),
  @conferences_days_id int
  as
  begin
    set nocount on
    begin try
      if not exists
        (
          select *
          from workshops
          where id = @id
        )
        begin
          throw 70000, 'There is no such workshop in the database', 1;
        end
      if not exists
        (
          select *
          from conferences_days
          where conferences_days.id = @conferences_days_id
        )
        begin
          ; throw 70000, 'There is no such conference day', 1
        end
      update workshops
      set name             = @name,
          time_start       = @time_start,
          time_end         = @time_end,
          max_participants = @max_participants,
          price            = @price
      where id = @id;
    end try
    begin catch
      declare @errorMessage nvarchar(1024) = 'Cannot update workshop ' +
          ERROR_MESSAGE();
      ; throw 70000, @errorMessage, 1
    end catch
  end
go
```

# 5 Funkcje

Wykaz funkcji, które można wykorzystywać korzystając z bazy danych

## 5.1 Funkcja doWorkshopsOverlapEachOther

Funkcja, która zwraca informację na temat tego, czy 2 podane warsztaty nachodzą na siebie w czasie

Listing 55: Funkcja doWorkshopsOverlapEachOther

```sql
create function doWorkshopsOverlapEachOther(@workshop_id_1 int, @workshop_id_2 int)
  returns bit
as
begin
  declare @start_1 time = dbo.startTimeOfWorkshop(@workshop_id_1)
  declare @end_1 time = dbo.endTimeOfWorkshop(@workshop_id_1)
  declare @start_2 time = dbo.startTimeOfWorkshop(@workshop_id_2)
  declare @end_2 time = dbo.endTimeOfWorkshop(@workshop_id_2)
  if @start_1 < @start_2 and @start_2 < @end_1 return 1
  if @start_2 < @start_1 and @start_1 < @end_2 return 1
  if @start_1 <= @start_2 and @end_2 <= @end_1 return 1
  if @start_2 <= @start_1 and @end_1 <= @end_2 return 1
  return 0
end
go
```

## 5.2 Funkcja endTimeOfWorkshop

Funkcja, która zwraca czas zakończenia danego warsztatu

Listing 56: Funkcja endTimeOfWorkshop

```
create function endTimeOfWorkshop(@workshop_id int)
  returns time(0)
as
begin
  return
    (
      select time_end
      from workshops w
      where @workshop_id = w.id
    )
end
go
```

## 5.3 Funkcja getPaidAmountByClient

Funkcja, która zwraca wartość opłaty wykonanej przez klienta

Listing 57: Funkcja getPaidAmountByClient

```
create function getPaidAmountByClient(@clientId int)
  returns float
as
begin
  return (
    select sum(p.pay_amount) total_paid
    from clients c
           join conferences_days_bookings cdb on c.id = cdb.clients_id
           join payments p on cdb.id = p.conferences_days_bookings_id
    where c.id = @clientId)
end
go
```

## 5.4 Funkcja getPriceOfConferencesDayByConferencesDayBookingId

Funkcja, która zwraca cene dnia konferencji dla konkretnego id

Listing 58: Funkcja getPriceOfConferencesDayByConferencesDayBookingId

```
create function getPriceOfConferencesDayByConferencesDayBookingId(@cdbID int)
  returns float
as
begin
  return (
    select top 1 pl.price
    from conferences_days_bookings cdb
           join conferences_days cd on cdb.conferences_days_id = cd.id
           join conferences c on cd.conferences_id = c.id
           join price_levels pl on c.id = pl.conferences_id
    where pl.date_end > cdb.booking_date
      and cdb.id = @cdbID
    order by pl.date_end ASC
  )
end
go
```

## 5.5 Funkcja getStudentDiscount

Funkcja, która zwraca cene zniżkę studencką

Listing 59: Funkcja getStudentDiscount

```
create function getStudentDiscount()
returns float
as
begin
  return (0.5)
end
go
```

## 5.6 Funkcja getWorkshopPriceById

Funkcja, która zwraca cene za warsztat dla danego id

Listing 60: Funkcja getWorkshopPriceById

```
create function getWorkshopPriceById(@id int)
  returns float
as
begin
  return (
    select price
    from workshops
    where id = @id
  )
end
go
```

## 5.7 Funkcja listOfWorkshopsForClient

Funkcja, która zwraca lista warsztatów dla danego klienta

Listing 61: Funkcja listOfWorkshopsForClient

```
CREATE function listOfWorkshopsForClient(@client_id int)
  returns table
    as
    return
          (
            select w.name, w.id
            from workshops w
                   join workshops_booking wb on w.id = wb.workshops_id
                   join conferences_days_bookings cdb on
                       wb.conferences_days_bookings_id = cdb.id
                   join clients c on cdb.clients_id = c.id
            where c.id = @client_id
          )
go
```

## 5.8 Funkcja listOfWorkshopsForClient

Funkcja, która zwraca czas rozpoczęcia danego warsztatu

Listing 62: Funkcja listOfWorkshopsForClient

```
create function startTimeOfWorkshop(@workshop_id int)
  returns time(0)
as
begin
  return (
    select time_start
    from workshops w
    where @workshop_id = w.id
  )
end
go
```

# 6 Triggery

Wykaz triggerów, które odpowiadają za dane w bazie

## 6.1 Trigger checkDataForClient

Trigger, który sprawdza poprawność danych w tabeli clients

Listing 63: Trigger checkDataForClient

```
CREATE trigger checkDataForClient
  on clients
  after insert, update
  as
begin
  set nocount on
  if exists
    (
      select *
      from inserted
             cross join clients
      where inserted.id <> clients.id
        and inserted.email = clients.email
    )
    begin
      ; throw 70031, 'There is already such an email in the database', 1
    end
  if exists
    (
      select *
      from inserted
             cross join clients
      where inserted.id <> clients.id
        and inserted.login = clients.login
    )
    begin
      ; throw 70032, 'There is already such a login in the database', 1
    end
end
go
```

## 6.2 Trigger checkDataForCompany

Trigger, który sprawdza poprawność danych w tabeli company

Listing 64: Trigger checkDataForCompany

```
CREATE trigger checkDataForCompany
  on company
  after insert, update
  as
begin
  set nocount on
  if exists
    (
      select *
      from inserted
             cross join company
      where inserted.id <> company.id
        and inserted.clients_id = company.clients_id
    )
    begin
      ; throw 70033, 'There is client with such id in the database', 1
    end
  if exists
    (
      select *
      from inserted
             cross join company
      where inserted.id <> company.id
        and inserted.NIP = company.NIP
    )
    begin
      ; throw 70033, 'There is client with such NIP in the database', 1
    end
end
go
```

## 6.3 Trigger blockDayThatExceedsConference

Trigger, który blokuje wstawienie nowego dnia konferencji do tabeli gdy jego data wykracza poza ramy czasowe samej konferencji

Listing 65: Trigger blockDayThatExceedsConference

```
CREATE trigger blockDayThatExceedsConference
  on conferences_days
  after insert, update
  as
  begin
    if exists
      (
        select *
        from conferences as c
               join conferences_days as cd on c.id = cd.conferences_id
        where cd.date < c.date_start
          or cd.date > c.date_end
      )
      begin
        ; throw 70023, 'Conference day exceeds the range of conference', 1
      end
  end
go
```

## 6.4 Trigger checkDateForConferenceDay

Trigger, który blokuje wstawienie nowego dnia konferencji do tabeli gdy istnieje już dzień konferencji z tą samą datą

Listing 66: Trigger checkDateForConferenceDay

```
create trigger checkDateForConferenceDay
  on conferences_days
  after insert, update
  as
  begin
    set nocount on
    if exists
      (
        select *
        from inserted
               cross join conferences_days
        where inserted.id <> conferences_days.id
          and inserted.date = conferences_days.date
          and inserted.conferences_id = conferences_days.conferences_id
      )
      begin
        ; throw 70034, 'There is already conference day with the same date', 1
      end
  end
go
```

## 6.5 Trigger checkBookingDateForConferenceDay

Trigger, który blokuje wstawienie rezerwacji, gdy zaistnieje sytuacja gdy konferencja jest rezerwowana mniej niz 14 dni przed jej rozpoczęciem

Listing 67: Trigger checkBookingDateForConferenceDay

```
CREATE trigger checkBookingDateForConferenceDay
  on conferences_days_bookings
  after insert, update
  as
begin
  if exists
    (
      select date
      from conferences_days cd
            left join conferences_days_bookings cdb on cd.id = cdb.conferences_days_id
      where date < dateadd(day, 14, cdb.booking_date)
    )
    begin
      ; throw 70024, 'Conference cannot be booked in less than 14 days before its
          start', 1
    end
end
go
```

## 6.6 Trigger notEnoughPlacesForConferenceDay

Trigger, który blokuje wstawienie nowego dnia konferencji w sytuacji gdy łączna liczba uczestników wszystkich dni konferencji przekracza maksymalną możliwą ilość uczestnikóœ całej konferencji

Listing 68: Trigger notEnoughPlacesForConferenceDay

```
CREATE trigger notEnoughPlacesForConferenceDay
  on conferences_days_bookings
  after insert, update
  as
begin
  set nocount on
  if exists
    (
      select *
      from conferences_days as cd
             left join conferences_days_bookings as cdbA on cd.id =
                 cdbA.conferences_days_id
      where (select sum(cdbB.booking_places)
             from conferences_days_bookings cdbB
             where cdbB.conferences_days_id = cd.id group by cdbB.booking_places) >
                 cd.max_participants
    )
    begin
      ; throw 70020, 'Not enough available places for conference day', 1;
    end
end
go
```

## 6.7 Trigger checkDataForParticipant

Trigger, który sprawdza poprawność danych w tabeli participant

Listing 69: Trigger checkDataForParticipant

```
create trigger checkDataForParticipant
  on participants
  after insert, update
  as
begin
  set nocount on
  if exists
    (
      select *
      from inserted
             cross join participants
      where inserted.id <> participants.id
        and inserted.email = participants.email
    )
  begin
    ; throw 70042, 'Email already exists in participants table', 1
  end
end
go
```

## 6.8 Trigger checkDataForPayment

Trigger, który sprawdza poprawność danych w tabeli payments

Listing 70: Trigger checkDataForPayment

```
CREATE trigger checkDataForPayment
  on payments
  after insert, update
  as
begin
  set nocount on
  if not exists
    (
      select *
      from inserted
             cross join conferences_days_bookings
      where inserted.conferences_days_bookings_id = conferences_days_bookings.id
    )
    begin
      ; throw 70040, 'Such conference day booking does not exist ', 1
    end
  if exists
    (
      select *
      from inserted
             cross join conferences_days_bookings
      where inserted.payment_date >= (select cancel_date
                                      from conferences_days_bookings
                                      where conferences_days_bookings.id =
                                          inserted.conferences_days_bookings_id)
    )
    begin
      ; throw 70041, 'Payment cannot be added to cancelled booking', 1
    end
end
go
```

### 6.9 Trigger checkPaymentDate

Trigger, który sprawdza czy konferencja została opłacona w odpowiednim terminie

Listing 71: Trigger checkPaymentDate

```
CREATE trigger checkPaymentDate
  on payments
  after insert, update
  as
begin
  if exists
    (
      select payment_date
      from payments as p
             left join conferences_days_bookings cdb on p.conferences_days_bookings_id
                 = cdb.id
             left join conferences_days_registrations cdr on cdb.id =
                 cdr.conferences_days_bookings_id
      where p.payment_date > dateadd(day, 7, cdr.registration_date)
    )
    begin
      ; throw 70025, 'Conference has to be paid in 7 days since registration', 1
    end
end
go
```

## 6.10 Trigger checkDataForPrizeLevel

Trigger, który sprawdza poprawność danych w tabeli prize$_levels$

Listing 72: Trigger checkDataForPrizeLevel

```
CREATE trigger checkDataForPrizeLevel
  on price_levels
  after insert, update
  as
begin
  set nocount on
  if not exists
    (
      select *
      from inserted
             cross join conferences
      where inserted.conferences_id = conferences.id
    )
    begin
      ; throw 70043, 'There is no such conference in the database', 1
    end
  if exists
    (
      select *
      from inserted
             cross join price_levels
      where inserted.date_end = price_levels.date_end
        and inserted.id != price_levels.id
        and inserted.conferences_id = price_levels.conferences_id
    )
  begin
    ; throw 70044, 'Duplicate end date for the prize level', 1
  end
end
go
```

## 6.11 Trigger checkDataForWorkshop

Trigger, który sprawdza poprawność danych w tabeli workshops

Listing 73: Trigger checkDataForWorkshop

```
create trigger checkDataForWorkshop
  on workshops
  after insert, update
  as
begin
  set nocount on
  if not exists
    (
      select *
      from inserted
             cross join conferences_days
      where inserted.conferences_days_id = conferences_days.id
    )
  begin
    ; throw 70045, 'There is no such conference day', 1
  end
end
go
```

## 6.12 Trigger checkDataForWorkshopBooking

Trigger, który sprawdza poprawność danych w tabeli workshops$_booking$

Listing 74: Trigger checkDataForWorkshopBooking

```
create trigger checkDataForWorkshopBooking
  on workshops_booking
  after insert, update as
begin
  set nocount on
  if not exists
    (
      select *
      from inserted
             cross join workshops
      where inserted.workshops_id = workshops.id
    )
    begin
      ; throw 70046, 'Workshop does not exist', 1
    end
  if not exists
    (
      select *
      from inserted
             cross join conferences_days_bookings
      where inserted.conferences_days_bookings_id = conferences_days_bookings.id
    )
    begin
      ; throw 70047, 'Conference day booking does not exist', 1
    end
end
go
```

## 6.13 Trigger morePlacesBookedForWorkshopThanForConferenceDay

Trigger, który blokuje wstawienie do tabeli workshops$_bookingwsytuacjigdyjestprbarezerwacjiwikszejilocimie$

Listing 75: Trigger morePlacesBookedForWorkshopThanForConferenceDay

```
CREATE trigger morePlacesBookedForWorkshopThanForConferenceDay
  on workshops_booking
  after insert, update
  as
begin
  set nocount on
  if exists
    (
      select *
      from workshops_booking as wb
             left join conferences_days_bookings cdb on
                 wb.conferences_days_bookings_id = cdb.id
      where (select sum(wb2.booking_place) from workshops_booking wb2 where
          wb2.conferences_days_bookings_id = cdb.id) >
            (select sum(cdb2.booking_places) from conferences_days_bookings cdb2 where
                cdb2.id = wb.conferences_days_bookings_id)
    )
    begin
      ; throw 70030, 'More places booked for the workshop than for the conference
          day', 1
    end
end
go
```

## 6.14 Trigger notEnoughPlacesForWorkshop

Trigger, który nie pozwala na wstawienie danych do tabeli workshops$_bookingwsytuacji, gdyjestzamaomiejscad$

Listing 76: Trigger notEnoughPlacesForWorkshop

```
CREATE trigger notEnoughPlacesForWorkshop
  on workshops_booking
  after insert, update
  as
begin
  set nocount on
  if exists
    (
      select *
      from workshops as w
             left join workshops_booking wb on w.id = wb.workshops_id
      where (select sum(wb2.booking_place) from workshops_booking wb2 where
          wb2.workshops_id = w.id) > w.max_participants
    )
    begin
      ; throw 70021, 'Not enough available places for workshop', 1
    end
end
go
```

## 6.15 Trigger tooManyStudents

Trigger, który nie pozwala na wstawienie danych do tabeli workshops_booking w sytuacji, gdy jest za mało miejsca dla warsztatu

Listing 77: Trigger tooManyStudents

```
create trigger tooManyStudents
  on workshops_booking
  after insert, update
  as
begin
  set nocount on
  if exists
    (
      select *
      from workshops_booking wb
      where wb.booking_place < wb.how_students
    )
    begin
      ; throw 70021, 'Too many students', 1
    end
end
go
```

## 6.16 Trigger checkMaxBookingPlaces

Trigger, który nie pozwala na wstawienie danych do tabeli workshops_registrations w sytuacji, gdy brakuje miejsca na warsztat

Listing 78: Trigger checkMaxBookingPlaces

```
CREATE trigger checkMaxBookingPlaces
  on workshops_registrations
  after insert, update
  as
  begin
    set nocount on
    if (select count(*)
        from workshops_registrations wr
        where wr.workshops_booking_id = (select workshops_booking_id from inserted)) >
            (
         select wb.booking_place
         from workshops_booking wb
         where wb.id = (select workshops_booking_id from inserted)
       )
      begin
        ; throw 70100, 'No more places', 4
      end
  end
go
```

## 6.17 Trigger checkWhetherClientIsAlreadyBookedForAnotherWorkshop

Trigger, który blokuje zapisanie się na warsztat w sytuacji, gdy uczestnik jest zapisany na inny warsztat trwający w tym samym czasie

Listing 79: Trigger checkWhetherClientIsAlreadyBookedForAnotherWorkshop

```
CREATE trigger checkWhetherClientIsAlreadyBookedForAnotherWorkshop
  on workshops_registrations
  after insert, update as
begin
  if exists
    (
    select *
from (select cd.id, w.time_start, w.time_end
      from participants p
             join workshops_registrations wr on wr.participants_id = p.id
             join workshops_booking wb on wb.id = wr.workshops_booking_id
             join workshops w on wb.workshops_id = w.id
             join conferences_days cd on w.conferences_days_id = cd.id
      where participants_id = (select i.participants_id from inserted i)) a
       inner join
     (select cd.id, w.time_start, w.time_end
      from participants p
             join workshops_registrations wr on wr.participants_id = p.id
             join workshops_booking wb on wb.id = wr.workshops_booking_id
             join workshops w on wb.workshops_id = w.id
             join conferences_days cd on w.conferences_days_id = cd.id
      where participants_id = (select i.participants_id from inserted i)) b on a.id =
          b.id and
                                      (a.time_start between b.time_start and
                                          b.time_end)
       and (a.time_end between b.time_start and b.time_end)
    )
    begin
      ; throw 70110, 'Client is already booked for another workshop', 5
    end
end
go
```

## 6.18 Trigger checkWorkshopRegistration

Trigger, który sprawdza poprawność wstawienia danych do tabeli workshops_registrations

Listing 80: Trigger checkWorkshopRegistration

```
create trigger checkWorkshopRegistration
  on workshops_registrations
  after insert, update
  as
  begin
    set nocount on
    if not exists
      (
        select *
        from inserted i
               join participants p on p.id = i.participants_id
               join conferences_days_registrations cdr on p.id = cdr.participants_id
        where cdr.conferences_days_bookings_id
          =
              (select wb.conferences_days_bookings_id
               from inserted i2
                      join workshops_booking wb on i2.workshops_booking_id = wb.id)
      )
      begin
        ; throw 70100, 'The data for workshops_registrations table is not valid', 4
      end
  end
go
```

### 6.19 Trigger participantExistsInWorkshop

Trigger, który nie pozwala na wstawienie danych do tabeli workshops_registrations w sytuacji gdy uczestnik już jest przypisany do warsztatu

Listing 81: Trigger participantExistsInWorkshop

```
create trigger participantExistsInWorkshop
  on workshops_registrations
  after insert, update
  as
begin
  set nocount on
  if (select count(*)
      from workshops_registrations wr
      where wr.workshops_booking_id = (select workshops_booking_id from inserted)
        and wr.participants_id =
            (select participants_id from inserted)) > 1
    begin
      ; throw 70100, 'Participant is already registered to this workshop', 4
    end
end
go
```

# 7 Indeksy

Po przeanalizowaniu użyywanych procedur i widoków zdecydowaliśmy założyć indeksy na kolumny po których najcześciej wyszukiwaliśmy dane.

## 7.1 Indeksy dla tabeli clients

```
create unique index clients_email_uindex
  on clients (email)
go

create unique index clients_account_number_uindex
  on clients (account_number)
go

create unique index clients_phone_number_uindex
  on clients (phone_number)
go
```

## 7.2 Indeksy dla tabeli company

```
create unique index company_NIP_uindex
  on company (NIP)
go

create unique index company_clients_id_uindex
  on company (clients_id)
go

create index company_client_id_index
  on company (clients_id)
go
```

## 7.3 Indeksy dla tabeli conferences

```
create index conferences_date_start_index
  on conferences (date_start)
go

create index conferences_date_end_index
  on conferences (date_end)
go
```

### 7.4 Indeksy dla tabeli conferences_days

```
create index conferences_day_conferences_id_index
  on conferences_days (conferences_id)
go

create index conferences_day_date_index
  on conferences_days (date)
go
```

### 7.5 Indeksy dla tabeli conferences_days_bookings

```
create index conferences_day_booking_conferences_day_id_index
  on conferences_days_bookings (conferences_days_id)
go

create index conferences_day_booking_clients_id_index
  on conferences_days_bookings (clients_id)
go
```

### 7.6 Indeksy dla tabeli conferences_days_registrations

```
create index conferences_day_registrations_conferences_day_booking_id_index
  on conferences_days_registrations (conferences_days_bookings_id)
go

create index conferences_day_registrations_participants_id_index
  on conferences_days_registrations (participants_id)
go

create index conferences_day_registrations_registration_date_index
  on conferences_days_registrations (registration_date)
go
```

### 7.7 Indeksy dla tabeli participants

```
create index participants_get_index
  on participants (firstname, surname, email)
go
```

## 7.8 Indeksy dla tabeli payments

```
create index payments_conferences_day_id_index
  on payments (conferences_days_bookings_id)
go

create index payments_payments_date_index
  on payments (payment_date)
go
```

## 7.9 Indeksy dla tabeli price_levels

```
create index price_levels_conferences_id_index
  on price_levels (conferences_id)
go

create index price_levels_date_end_index
  on price_levels (date_end)
go
```

## 7.10 Indeksy dla tabeli workshops

```
create index workshop_conferences_day_id_index
  on workshops (conferences_days_id)
go
```

## 7.11 Indeksy dla tabeli workshops_booking

```
create index workshop_booking_workshop_id_index
  on workshops_booking (workshops_id)
go

create index workshop_booking_conferences_day_booking_id_index
  on workshops_booking (conferences_days_bookings_id)
go
```

# 8 Role

Propozycja roli w bazie dancyh

## 8.1 Administrator

Administratorem jest osoba która zarządza systemem, posiada należyte umiejętności w posługiwaniu się językiem SQL oraz zarządzaniem bazą danych. Posiada dostęp do wszystkich elementów bazy danych, widoków i procedur.

## 8.2 Pracownik firmy

Przez pracownika firmy rozumie się osobę zatrudnioną w firmie, która obsługuje konferencje oraz zajmuje się kontaktem z klientami. Powinien mieć możliwość odczytu i zapisu danych, natomiast nie może edystować wydoków czy proceur.

## 8.3 Klient

Kliet powinien mieć dostęp do składania rezerwacji i rejestracji miejsc na konferencje.

## 9 Generator danych

Aby wygenerować spójne i sensowne dane napisaliśmy generator w Pythonie powniewarz udostępnia on przydatne narzędzia przy tworzeniu tego typu algorytmów.

Listing 82: Kod generatora danych

```
import os
from os import listdir
from os.path import isfile, join
import random as random
import pymssql
from datetime import datetime as dt
from datetime import timedelta
import re
import sys
from random import randrange

class Generator(object):

    server = "dbadmin.iisg.agh.edu.pl"
    user = 'piaskowy'
    password = '########'
    dbName = 'piaskowy_a'
    connection = 0
    db = 0

    def __init__(self):
        self.db = pymssql.connect(self.server, self.user, self.password, self.dbName)

    def __del__(self):
        self.db.close()

    def genConferences(self):
        how = 100
        conferencesNameFile = open('data/conferences_name.txt', 'r')
        conferencesName = conferencesNameFile.readlines()
        conferencesPlaceFile = open('data/conferences_place.txt', 'r')
        conferencesPlace = conferencesPlaceFile.readlines()

        out = []
        fileName = 'clientsSql.sql'
        if os.path.exists(fileName):
            os.remove(fileName)
        sqlOutputs = open(fileName, 'a')

        i = 0
        while i < how:
            adress = self.getRandElementFromTable(conferencesPlace)
            adress = adress.split(',')
            name = self.getRandElementFromTable(conferencesName)
            date = self.randomDate(dt.strptime('2016-01-01', '%Y-%m-%d'),
                dt.strptime('2020-12-12', '%Y-%m-%d'))
            dateArray = str(date).split(' ')
            nextDate = self.randomNextDate(date)
            nextDateArray = str(nextDate).split(' ')
            query = 'exec addNewConference \
            @name = "' + name + '", \
            @date_start = "' + str(dateArray[0]) + '", \
            @date_end = "' + str(nextDateArray[0]) + '", \
            @city = "' + str(adress[0]) + '", \
            @street = "' + str(adress[1]) + '", \
            @local_number = "' + str(adress[2]) + '", \
            @zip = "' + str(adress[3]) + '";'
            if query in out:
                continue
            out.append(query)
            i += 1

        for i in out:
            sqlOutputs.write(self.cleanItem(i) + '\n')
        sqlOutputs.close()

    def getConferencesDay(self):
        themesSet = open('data/conferences_theme.txt', 'r').readlines()

        if os.path.exists('confDay.sql'):
            os.remove('confDay.sql')
        sqlOutputs = open('confDay.sql', 'a')
        out = []
        allConf = self.db.cursor(as_dict=True)
        allConf.execute("select * from conferences")

        for conf in allConf:
            i = 0
            dayDate = dt.strptime(conf['date_start'], '%Y-%m-%d')
            date_end = dt.strptime(conf['date_end'], '%Y-%m-%d')
            while dayDate <= date_end:
                query = 'exec addNewConferenceDay \
                @conferences_id = '+ str(conf['id']) + ', \
                @theme = "' + self.getRandElementFromTable(themesSet) + '", \
                @date = "' + str(dayDate)[:10] + '", \
                @time_start = "' + self.getRandTimeStart() + '", \
                @time_end = "' + self.getRandTimeEnd() + '", \
                @max_participants = ' + str(random.randint(16, 300))
                out.append(query)
                dayDate += timedelta(days=1)

        for i in out:
            sqlOutputs.write(self.cleanItem(i) + '\n')
        sqlOutputs.close()

    def genClients(self, how):
        namesFile = open('data/names.txt', 'r')
        namesSet = namesFile.readlines()
        surnamesFile = open('data/surnames.txt', 'r')
        surnamesSet = surnamesFile.readlines()
        placesFile = open('data/place.txt', 'r')
        places = placesFile.readlines()
    out = []
    sqlOutputs = open('clientsSql.sql', 'a')
    emailSet = [] #email must be unique
    phoneSet = [] #phone must be unique
    loginSet = [] #login must be unique
    acNoSet = [] #account number must be unique

    i = 0
    while i < how:
        name = self.getRandElementFromTable(namesSet)
        surname = self.getRandElementFromTable(surnamesSet)
        adress = self.getRandElementFromTable(places)
        adress = adress.split(',')
        nr = adress[2]
        nr = nr.split('/')

        email = name + '.' + surname + '@gmail.com'
        phone = random.randint(528349553, 967942954)
        login = name + surname + str(random.randint(0, 99999))
        pasHash = self.getHash()
        accountNumber = self.getAccountNumber()
        bit = random.randint(0, 1)

        if email in emailSet or phone in emailSet or login in loginSet:
            continue

        query = 'INSERT INTO `clients` (name, phone_number, email, password_hash,
            account_number, city, street, house_number, flat_number, zip,
            is_company) values ("' \
        + name + ' ' + surname + '", "' \
        + str(phone) + '", "' \
        + email + '", "' \
        + login + '", "' \
        + pasHash + '", "' \
        + accountNumber + '", "' \
        + adress[0] + '", "' \
        + adress[1] + '", "' \
        + nr[0] + '", "' \
        + nr[1] + '", "' \
        + adress[3] + '", ' \
        + str(bit) + ')'

        if query in out:
            continue
        out.append(query)
        emailSet.append(email)
        phoneSet.append(phone)
        loginSet.append(login)
        i += 1

    for i in out:
        sqlOutputs.write(i + '\n')
    sqlOutputs.close()

def getWorkshop(self):
    themesSet = open('data/conferences_theme.txt', 'r').readlines()
    fileName = 'workshop.sql'
    if os.path.exists(fileName):
        os.remove(fileName)
    sqlOutputs = open(fileName, 'a')
    out = []
    allConfDay = self.db.cursor(as_dict=True)
    allConfDay.execute("select * from conferences_days")

    for confDay in allConfDay:
        i = 0
        howWorkshop = random.randint(1, 3)
        while i < howWorkshop:
            query = 'exec addNewWorkshop \
            @name = "' + self.getRandElementFromTable(themesSet) + '", \
            @time_start = "' + self.getRandTimeStartWorkshop() + '", \
            @time_end = "' + self.getRandTimeEndWorkshop() + '", \
            @max_participants = ' + str(random.randint(16,
                confDay['max_participants'])) + ', \
            @price = "' + str(random.randint(1, 200)) + '.' +
                str(random.randint(0, 100)) + '", \
            @conferences_days_id = '+ str(confDay['id'])
            out.append(query)
            i += 1

    for i in out:
        sqlOutputs.write(self.cleanItem(i) + '\n')
    sqlOutputs.close()

def getPriceLevels(self):
    fileName = 'pricelevel.sql'
    if os.path.exists(fileName):
        os.remove(fileName)
    sqlOutputs = open(fileName, 'a')
    out = []
    allConf = self.db.cursor(as_dict=True)
    allConf.execute("select * from conferences")

    for conf in allConf:
        i = 0
        howPriceLevel = random.randint(1, 4)
        dateLvl = dt.strptime(conf['date_start'], '%Y-%m-%d')
        startPrice = random.randint(50, 350)
        while i < howPriceLevel:
            dateEnd = str(dateLvl).split(' ')[0]
            query = 'exec addNewPriceLevel \
            @price = "' + str(startPrice) + '", \
            @date_end = "' + dateEnd + '", \
            @conferences_id = ' + str(conf['id'])
            startPrice *= 0.7
            dateLvl -= timedelta(days=random.randint(1, 10))
            out.append(query)
            i += 1

    for i in out:
        sqlOutputs.write(self.cleanItem(i) + '\n')
    sqlOutputs.close()

def getConferencessDayBooking(self):
    fileName = 'conferencesDayBooking.sql'
    if os.path.exists(fileName):
        os.remove(fileName)
    sqlOutputs = open(fileName, 'a')
    out = []
    allClients = self.db.cursor(as_dict=True)
    allClients.execute("select * from clients")
    clientsList = allClients.fetchall()
    allConfDay = self.db.cursor(as_dict=True)
    allConfDay.execute("select * from conferences_days")
    for confDay in allConfDay:
        confDayDate = dt.strptime(confDay['date'], '%Y-%m-%d')
        maxBookPlace = int(confDay['max_participants']*0.25)-1
        if maxBookPlace < 1:
            maxBookPlace = 1
        bookingPlace = random.randint(1, maxBookPlace)
        maxRand = random.randint(1, 3)
        i = 0
        while i < maxRand:
            query = 'exec addNewConferenceDaysBooking \
            @conferences_days_id = ' + str(confDay['id']) + ', \
            @booking_date = "' + str(confDayDate -
                timedelta(days=random.randint(15, 70))).split(' ')[0] + '", \
            @booking_places = ' + str(bookingPlace) + ',\
            @how_students = ' + str(bookingPlace * (1/random.randint(1, 100))) +
                ', \
            @clients_id = ' +
                str(self.getRandElementFromTableStandard(clientsList)['id']) + ', \
            @cancel_date = null'
            out.append(query)
            i += 1

    for i in out:
        sqlOutputs.write(self.cleanItem(i) + '\n')
    sqlOutputs.close()

def getWorkshopBooking(self):
    fileName = 'workshopBooking.sql'
    if os.path.exists(fileName):
        os.remove(fileName)
    sqlOutputs = open(fileName, 'a')
    out = []
    allConferencesDayBooking = self.db.cursor(as_dict=True)
    allConferencesDayBooking.execute("select * from conferences_days_bookings")
    allConferencesDayBookingList = allConferencesDayBooking.fetchall()
    allWorkshop = self.db.cursor(as_dict=True)
    allWorkshop.execute("select * from workshops")

    for workshop in allWorkshop:
        maxRand = random.randint(1, 3)
        i = 0
        while i < maxRand:
            bookedInThisTour = 0
            idCDB = self.existRelation(workshop['conferences_days_id'],
                'conferences_days_id', allConferencesDayBookingList);
            if idCDB > 0:
                item = self.getItemById(idCDB, allConferencesDayBookingList)
                maxBookPlace = int(item['booking_places']*0.25)
                if maxBookPlace < 1:
                    maxBookPlace = 1
                bookingPlace = random.randint(1, maxBookPlace)
                bookedInThisTour += bookingPlace
                if bookedInThisTour > item['booking_places']:
                    break
                query = 'exec addNewWorkshopBooking \
                @workshop_id = ' + str(workshop['id']) + ', \
                @booking_place = ' + str(bookingPlace) + ', \
                @booking_date = "' + str(item['booking_date']) + '", \
                @conferences_day_booking_id = ' + str(idCDB) + ' \
                @how_students = ' + str(int(0.5 * bookingPlace))
                out.append(query)
            i += 1

    for i in out:
        sqlOutputs.write(self.cleanItem(i) + '\n')
    sqlOutputs.close()

def getParticipants(self):
    fileName = 'participants.sql'
    if os.path.exists(fileName):
        os.remove(fileName)
    sqlOutputs = open(fileName, 'a')

    namesFile = open('data/names.txt', 'r')
    namesSet = namesFile.readlines()
    surnamesFile = open('data/surnames.txt', 'r')
    surnamesSet = surnamesFile.readlines()

    emailSet = []

    out = []
    i = 0
    while i < 3000:
        name = self.getRandElementFromTable(namesSet)
        surname = self.getRandElementFromTable(surnamesSet)
        email = name + '.' + surname + '@gmail.com'
        if email in emailSet:
            continue
        query = 'exec addNewParticipant \
        @firstname = "' + name + '", \
        @surname = "' + surname + '", \
        @email = "' + email
        emailSet.append(email)
        out.append(query)
        i += 1
    for i in out:
        sqlOutputs.write(re.sub("            ", " ", i) + '\n')
    sqlOutputs.close()

def getConferencesDayReservation(self):
    fileName = 'conferencesDayReservation.sql'
    if os.path.exists(fileName):
        os.remove(fileName)
    sqlOutputs = open(fileName, 'a')
    out = []

    tmp = self.db.cursor(as_dict=True)
    tmp.execute("select count(*) as participantsCount from participants")
    for item in tmp:
        participantsCount = item['participantsCount']

    allParticipants = self.db.cursor(as_dict=True)
    allParticipants.execute("select * from participants")
    participantsList = allParticipants.fetchall()

    allConferencesDayBooking = self.db.cursor(as_dict=True)
    allConferencesDayBooking.execute("select * from conferences_days_bookings")

    for confDayBooking in allConferencesDayBooking:
        i = 0
        howStudents = 0;
        isStudent = 0
        participantsSet = []
        while i < confDayBooking['booking_places']:
            participant = self.getRandElementFromTableStandard(participantsList)
            if participant['id'] in participantsSet:
                continue
            if howStudents < confDayBooking['how_students']:
                isStudent = 1
                howStudents += 1
            else:
                isStudent = 0
            query = 'exec addNewConferenceDaysRegistration \
            @conferences_days_booking_id = ' + str(confDayBooking['id']) + ', \
            @participants_id = ' + str(participant['id']) + ',\
            @is_student = ' + str(isStudent) + ', \
            @registration_date = "' + str(confDayBooking['booking_date']) + '"'
            participantsSet.append(participant['id'])
            out.append(query)
            i += 1

    for i in out:
        sqlOutputs.write(self.cleanItem(i) + '\n')
    sqlOutputs.close()

def getCompany(self):
    fileName = 'company.sql'
    if os.path.exists(fileName):
        os.remove(fileName)
    sqlOutputs = open(fileName, 'a')
    out = []

    tmp = self.db.cursor(as_dict=True)
    tmp.execute("select count(*) as participantsCount from participants")
    participantsCount = tmp['participantsCount']

    allParticipants = self.db.cursor(as_dict=True)
    allParticipants.execute("select * from participants")
    participantsList = allParticipants.fetchall()

    allConferencesDayBooking = self.db.cursor(as_dict=True)
    allConferencesDayBooking.execute("select * from conferences_days_bookings")

    participantsSet = []

    for confDayBooking in allConferencesDayBooking:
        i = 0
        howStudents = 0;
        isStudent = 0
        while i < confDayBooking['booking_places']:
            participant = self.getRandElementFromTableStandard(participantsList)
            if participant['id'] in participantsSet:
                continue
            if howStudents < confDayBooking['how_students']:
                isStudent = 1
                howStudents += 1
            else:
                isStudent = 0
            query = 'exec addNewConferenceDaysRegistration \
            @conferences_days_booking_id = ' + confDayBooking['id'] + ', \
            @participants_id = ' + participant['id'] + ',\
            @is_student = ' + isStudent

            participantsSet.append(participant['id'])
            out.append(query)
            i += 1

    for i in out:
        sqlOutputs.write(re.sub("             ", " ", i) + '\n')
    sqlOutputs.close()

def getWorkshopReservation(self):
    fileName = 'workshopReservation.sql'
    if os.path.exists(fileName):
        os.remove(fileName)
    sqlOutputs = open(fileName, 'a')
    out = []

    allWorkshopBooking = self.db.cursor(as_dict=True)
    allWorkshopBooking.execute("select * from workshops_booking")
    allWorkshopBookingList = allWorkshopBooking.fetchall()

    for workshopBooking in allWorkshopBookingList:
        confDayReseg = self.db.cursor(as_dict=True)
        confDayReseg.execute("select * from conferences_days_registrations where
            conferences_days_bookings_id = " +
            str(workshopBooking['conferences_days_bookings_id']))

        i = 0
        for item in confDayReseg:
            if(i >= workshopBooking['booking_place']):
                break
            query = 'insert into workshops_registrations (workshops_booking_id,
                cancel_date, registration_date, participants_id) values (' +
                str(workshopBooking['id']) + ', null, \'' +
                str(workshopBooking['booking_date']) + '\', ' +
                str(item['participants_id']) + ');'
            out.append(query)
            i += 1

    for i in out:
        sqlOutputs.write(self.cleanItem(i) + '\n')
    sqlOutputs.close()

def getPayments(self):
    fileName = 'payments.sql'
    if os.path.exists(fileName):
        os.remove(fileName)
    sqlOutputs = open(fileName, 'a')
    out = []
    payMethod = ["cash", "card"]
    allBooking = self.db.cursor(as_dict=True)
    allBooking.execute("select * from conferences_days_bookings where cancel_date
        is null")

    for booking in allBooking:
        query = 'exec addNewPayment \
        @pay_amount = ' + str(random.randint(200, 1000)) + ".00" + ', \
        @conferences_days_booking_id = ' + str(booking['id']) + ', \
        @payment_date = "' + str(dt.strptime(booking['booking_date'], '%Y-%m-%d')
            + timedelta(days=random.randint(1, 5))).split(' ')[0] + '", \
        @payments_method = "' + payMethod[random.randint(0, 1)] + '"'
        out.append(query)

    for i in out:
        sqlOutputs.write(self.cleanItem(i) + '\n')
    sqlOutputs.close()

##################

def getItemById(selt, searchId, searchSet):
    for item in searchSet:
        if item['id'] == searchId:
            return item
    return null

def cleanItem(self, item):
    out = item
    while out.find("  ") > 0:
        out = re.sub("  ", " ", out)
return out

def existRelation(self, searchedValue, searchIndex, relationsSet):
for item in relationsSet:
    if item[searchIndex] == searchedValue:
        return item['id']
return 0

def execFile(self, path):
sqlQueries = open(path, 'r').readlines()
executor = self.db.cursor(as_dict=True)
allQuery = 0
errorQuery = 0
for query in sqlQueries:
    allQuery += 1
    print(allQuery)
    try:
        executor.execute(query)
    except:
        errorQuery += 1
        print("error: " + str(sys.exc_info()))
print("Error: " + str(errorQuery))
print("All: " + str(allQuery))

def getRandTimeStart(self):
return str(random.randint(7, 9)) + ':' + str(random.randint(0, 30)) + ":00"

def getRandTimeEnd(self):
return str(random.randint(10, 12)) + ':' + str(random.randint(0, 30)) + ":00"

def getRandTimeStartWorkshop(self):
return str(random.randint(13, 16)) + ':' + str(random.randint(0, 30)) + ":00"

def getRandTimeEndWorkshop(self):
return str(random.randint(17, 20)) + ':' + str(random.randint(0, 30)) + ":00"

def getRandElementFromTable(self, setArray):
return setArray[random.randint(0, len(setArray) - 1)].rstrip("\n\r")

def getRandElementFromTableStandard(self, setArray):
return setArray[random.randint(0, len(setArray) - 1)]

def randomDate(self, start, end):
delta = end - start
int_delta = (delta.days * 24 * 60 * 60) + delta.seconds
random_second = randrange(int_delta)
return start + timedelta(seconds=random_second)

def randomNextDate(self, date):
strNextDate = str(date + timedelta(days=random.randint(0, 5)))
dateArray = strNextDate.split(' ')
strOut = dt.strptime(dateArray[0], '%Y-%m-%d')
return strOut

def getHash(self):
    chars = "qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM1234567890"
    hashStr = ""
    for i in range(30):
        hashStr += chars[random.randint(0, len(chars) - 1)]
    return hashStr

def getAccountNumber(self):
    number = ""
    for i in range(24):
        number += str(random.randint(0, 9))
    return number
```