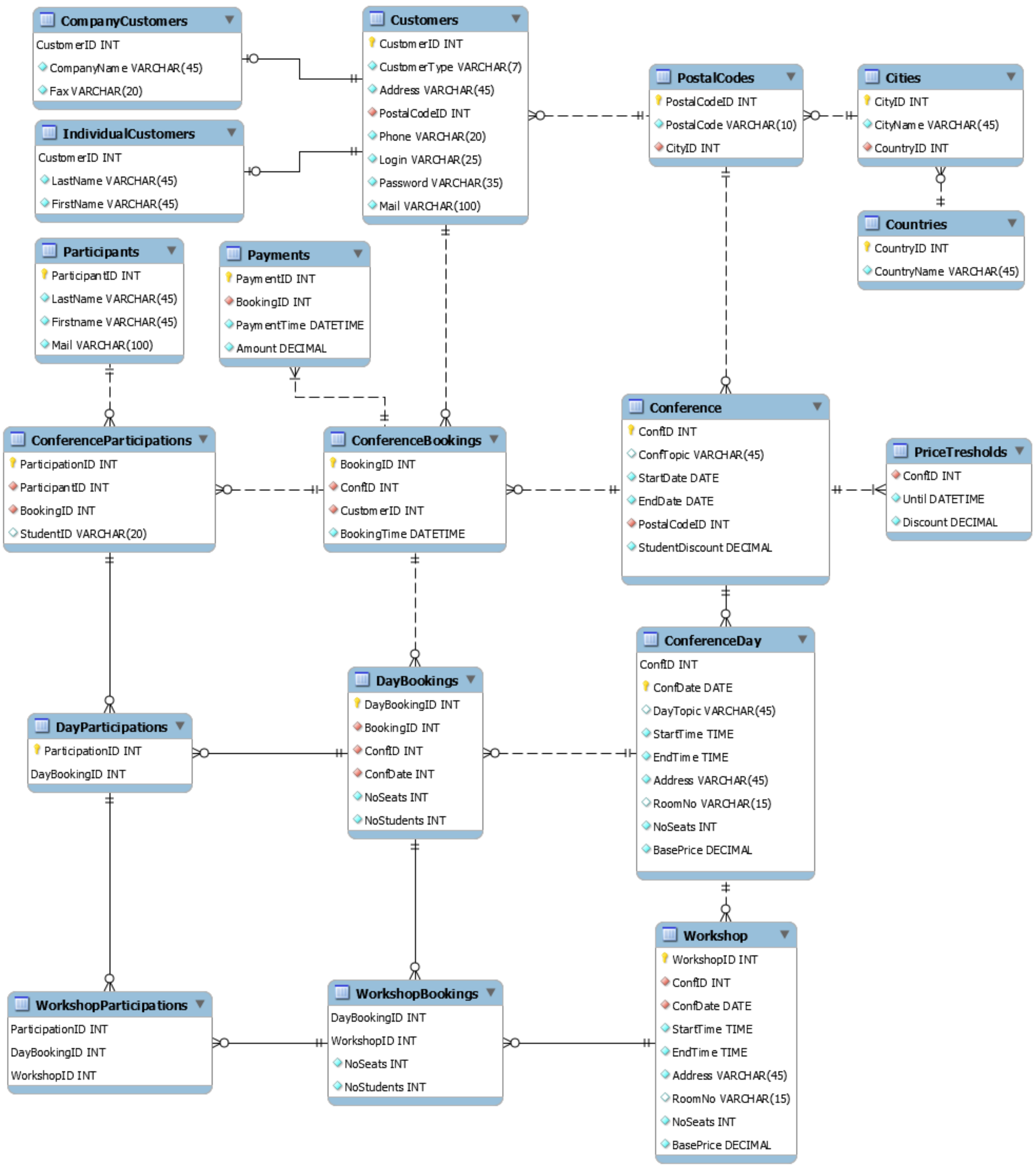
Spis treści

1 Opis systemu

System obsługuje bazę danych firmy organizującej konferencje. Każda konferencja może trwać jeden lub kilka dni. Klienci, którymi mogą być zarówno osoby indywidualne jak i firmy, rezerwują miejsca osobno na każdy dzień konferencji, a później dosyłają firmie listy uczestników. Klient ma także możliwość zmiany liczby uczestników lub anulowania całej rezerwacji do dwóch tygodni przed konferencją. Od czasu rezerwacji zależy zniżka procentowa. Organizowane są także warsztaty, w których mogą brać udział tylko osoby zapisane tego samego dnia na konferencję. Dla studentów przewidziane są zniżki. Głównym narzędziem służącym do komunikacji z systemem jest interfejs webowy, gdzie klienci logują się na swoje konta, a strona korzysta z API bazy aby pobierać i umieszczać dane w bazie.
Projekt został zaimplementowany w systemie **PostgreSQL**. Graficzny schemat bazy danych wykonano przy wykorzystaniu programu MySQL Workbench.

2 Diagram bazy danych



Rys. 1. Diagram zaprojektowanej bazy danych

3 Tabele

3.1 Countries

Tabela przechowująca nazwy państw.

```sql
CREATE TABLE IF NOT EXISTS Countries
(CountryID SERIAL PRIMARY KEY,
 CountryName VARCHAR(45) NOT NULL,
 UNIQUE(CountryName));
```

3.2 Cities

Tabela przechowująca miasta.

```sql
CREATE TABLE IF NOT EXISTS Cities
(CityID SERIAL PRIMARY KEY,
 CityName VARCHAR(45) NOT NULL,
 CountryID INT NOT NULL,
 FOREIGN KEY (CountryID) REFERENCES Countries);
```

3.3 PostalCodes

Tabela przechowująca kody pocztowe.

```sql
CREATE TABLE IF NOT EXISTS PostalCodes
(PostalCodeID SERIAL PRIMARY KEY,
 PostalCode VARCHAR(10) NOT NULL,
 CityID INT NOT NULL,
 UNIQUE(PostalCode, CityID),
 FOREIGN KEY (CityID) REFERENCES Cities);
```

3.4 Conference

Jest to tabela przechowująca informacje o organizowanych konferencjach: temat przewodni konferencji (ConfTopic), daty rozpoczęcia i zakończenia (StartDate i EndDate), lokalizację konferencji (City, PostalCode), długość trwania (NoDays), wysokość zniżki przysługującej studentom (StudentDiscount) wyrażoną poprzez liczbę z zakresu od 0 do 1.

```
CREATE TABLE IF NOT EXISTS Conference
(ConfID SERIAL PRIMARY KEY,
 ConfTopic VARCHAR(100) NULL,
 StartDate DATE NOT NULL,
 EndDate DATE NOT NULL,
 PostalCodeID INT NOT NULL,
 StudentDiscount DECIMAL NOT NULL,
 CHECK (StartDate <= EndDate),
 CHECK (StudentDiscount >= 0),
 FOREIGN KEY (PostalCodeID) REFERENCES PostalCodes);
```

3.5 Customers

Tabela, do której wpisywani są klienci. Klientem może być osoba prywatna lub firma, czemu odpowiadają odpowiednio wartości ‘Person’ i ‘Company’ atrybutu CustomerType. Oprócz tego tabela przechowuje atrybuty, które posiada każdy klient niezależnie od typu: dane adresowe (Address, City, Country, PostalCode), kontaktowe (Phone, Mail), dane logowania do naszego serwisu (Login, Password). Każdemu wpisowi w Customers ma odpowiadać dokładnie jeden wpis w IndividualCustomers lub CompanyCustomers.

```
CREATE TABLE IF NOT EXISTS Customers
(CustomerID SERIAL PRIMARY KEY,
 CustomerType VARCHAR(7) NOT NULL,
 Address VARCHAR(45) NOT NULL,
 PostalCodeID INT NOT NULL,
 Phone VARCHAR(16) NOT NULL,
 Login VARCHAR(25) NOT NULL,
 Password VARCHAR(35) NOT NULL,
 Mail VARCHAR(70) NOT NULL,
 UNIQUE (Phone),
 UNIQUE (login),
 UNIQUE (Mail),
 CHECK (is_valid_phone_or_fax(Phone)),
 CHECK (is_valid_login(Login)),
 CHECK (is_valid_mail(Mail)),
 CHECK (CustomerType IN('company','person')),
 FOREIGN KEY (PostalCodeID) REFERENCES PostalCodes);
```

3.6 CompanyCustomers

Tabela, do której wpisywani są klienci zbiorowi reprezentowani przez firmy. Składowane są tutaj takie dane jak: nazwa firmy (CompanyName), fax firmowy (Fax) oraz odwołanie do jej wpisu w Customers (CustomerID).

```
CREATE TABLE IF NOT EXISTS CompanyCustomers
(CustomerID SERIAL PRIMARY KEY,
 CompanyName VARCHAR(45) NOT NULL,
 Fax VARCHAR(16) NULL,
 CHECK (is_valid_phone_or_fax(Fax)),
 FOREIGN KEY (CustomerID) REFERENCES Customers);
```

3.7 IndividualCustomers

tabela, do której wpisywani są klienci będący osobami fizycznymi. Przechowuje imię i nazwisko klienta (FirstName, LastName) oraz odwołanie do jego wpisu w tabeli Customers (CustomerID).

```
CREATE TABLE IF NOT EXISTS IndividualCustomers
(CustomerID SERIAL PRIMARY KEY,
 LastName VARCHAR(45) NOT NULL,
 FirstName VARCHAR(45) NOT NULL,
 CHECK (is_valid_name(FirstName)),
 CHECK (is_valid_name(LastName)),
 FOREIGN KEY (CustomerID) REFERENCES Customers);
```

3.8 ConferenceBookings

Jest to tabela reprezentująca rezerwacje konferencji dokonane przez klientów. Informacje, które są tutaj magazynowane to: odwołanie do konferencji z tabeli Conferences (ConfID) i klientów z tabeli Customers (CustomerID), jak również czas złożenia rezerwacji (When).

```
CREATE TABLE IF NOT EXISTS ConferenceBookings
(BookingID SERIAL PRIMARY KEY,
 ConfID INT NOT NULL,
 CustomerID INT NOT NULL,
 BookingTime TIMESTAMP NOT NULL,
 UNIQUE (BookingID, ConfID),
 FOREIGN KEY (ConfID) REFERENCES Conference,
 FOREIGN KEY (CustomerID) REFERENCES Customers);
```

3.9 ConferenceDay

To tabela, w której znajdują się szczegółowe informacje o danym dniu konferencji. Dzień odwołuje się do konkretnej konferencji z tabeli Conferences korzystając z identyfikatora konferencji (ConfID). Znajdują się tutaj dane takie jak: data (ConfDate), czas rozpoczęcia i czas zakończenia konferencji (StartTime i EndTime), adres (Address), opcjonalnie numer pokoju (RoomNo). Tabela posiada również pola mówiące o ilości zarezerwowanych miejsc (NoSeats) oraz bazowej cenie za uczestnictwo (BasePrice).

```
CREATE TABLE IF NOT EXISTS ConferenceDay
(ConfID INT NOT NULL,
 ConfDate DATE NOT NULL,
 DayTopic VARCHAR(100) NULL,
 StartTime TIME NOT NULL,
 EndTime TIME NOT NULL,
 Address VARCHAR(45) NOT NULL,
 RoomNo VARCHAR(15) NULL,
 NoSeats INT NOT NULL,
 BasePrice DECIMAL NOT NULL,
 PRIMARY KEY (ConfID, ConfDate),
 CHECK (NoSeats > 0),
 CHECK (BasePrice >= 0),
 FOREIGN KEY (ConfID) REFERENCES Conference);
```

3.10 Workshop

Jest to tabela, która dostarcza nam wiadomości o organizowanych warsztatach podczas trwania danej konferencji, odwołując się do ConferenceDay (ConfID, ConfDate). Zapisywane są tutaj następujące informacje: godzina początku oraz zakończenia (StartTime, EndTime), adres odbywających się warsztatów (Address), opcjonalny numer pomieszczenia (RoomNo) oraz podstawowa cena udziału w warsztatach (BasePrice).

```
CREATE TABLE IF NOT EXISTS Workshop
(WorkshopID SERIAL PRIMARY KEY,
 ConfID INT NOT NULL,
 ConfDate DATE NOT NULL,
 WorkshopTopic VARCHAR(100) NULL,
 StartTime TIME NOT NULL,
 EndTime TIME NOT NULL,
 Address VARCHAR(45) NOT NULL,
 RoomNo VARCHAR(15) NULL,
 NoSeats INT NOT NULL,
 BasePrice DECIMAL NOT NULL,
 CHECK (EndTime > StartTime),
 CHECK (NoSeats > 0),
 CHECK (BasePrice >= 0),
 FOREIGN KEY (ConfID, ConfDate) REFERENCES ConferenceDay);
```

3.11 DayBookings

To tabela w której znajdują się rezerwacje miejsc na poszczególne dni podczas trwania danej konferencji. Rezerwacja dnia zawiera odwołanie do rezerwacji konferencji w ConferenceBookings (BookingID) oraz do dnia w ConferenceDay (ConfID i ConfDate), a także liczbę zgłoszonych uczestników (NoSeats) i liczbę znajdujących się wśród nich studentów (NoStudent), dla których koszt uczestnictwa jest mniejszy.

```
CREATE TABLE IF NOT EXISTS DayBookings
(DayBookingID SERIAL PRIMARY KEY,
 BookingID INT NOT NULL,
 ConfID INT NOT NULL,
 ConfDate DATE NOT NULL,
 NoSeats INT NOT NULL,
 NoStudents INT NOT NULL,
 CHECK (NoSeats > 0),
 CHECK (NoStudents >= 0),
 CHECK (NoStudents <= NoSeats),
 FOREIGN KEY (BookingID, ConfID) REFERENCES
  ↪ ConferenceBookings(BookingID, ConfID),
 FOREIGN KEY (ConfID, ConfDate) REFERENCES ConferenceDay);
```

3.12 WorkshopBookings

Tabela przedstawiająca rezerwacje na konkretne warsztaty. Każda z rezerwacji posiada odwołanie do warsztatu w Workshops (WorkshopID) i do rezerwacji dnia w DayBookings (DayBookingID) oraz informację o liczbie zarezerwowanych miejsc (NoSeats).

```sql
CREATE TABLE IF NOT EXISTS WorkshopBookings
(DayBookingID INT NOT NULL,
 WorkshopID INT NOT NULL,
 NoSeats INT NOT NULL,
 NoStudents INT NOT NULL,
 CHECK (NoSeats > 0),
 CHECK (NoStudents >= 0),
 CHECK (NoStudents <= NoSeats),
 PRIMARY KEY (DayBookingID, WorkshopID),
 FOREIGN KEY (DayBookingID) REFERENCES DayBookings,
 FOREIGN KEY (WorkshopID) REFERENCES Workshop);
```

3.13 Participants

Jest to tabela, która reprezentuje osoby – uczestników konferencji. Głównym atrybutem identyfikującym uczestnika jest jego mail. Poza tym przechowywane są także jego imię (FirstName) i nazwisko (LastName). Dany uczestnik NIE jest przypisany na stałe do klienta – wiele z przechowywanych w tabeli ConferenceParticipations uczestnictw w konferencji może się odwoływać do tego samego uczestnika nawet jeśli odpowiadają one rezerwacjom różnych klientów.

```sql
CREATE TABLE IF NOT EXISTS Participants
(ParticipantID SERIAL PRIMARY KEY,
 LastName VARCHAR(45) NULL,
 Firstname VARCHAR(45) NULL,
 Mail VARCHAR(70) NOT NULL,
 UNIQUE (Mail),
 CHECK (is_valid_name(FirstName)),
 CHECK (is_valid_name(LastName)),
 CHECK (is_valid_mail(Mail)));
```

3.14 ConferenceParticipations

Tabela opisująca uczestnictwo osób w danej konferencji. Odwołuje się do uczestników z Participants (ParticipantID), a także rezerwacji z ConferenceBookings (BookingID). Jeśli w kontekście danej konferencji uczestnik jest studentem,wówczas atrybut StudentID jest atrybutem służącym do przechowywania jego identyfikatora (nr legitymacji studenckiej). W przeciwnym wypadku pole StudentID przyjmuje wartość NULL.

```sql
CREATE TABLE IF NOT EXISTS ConferenceParticipations
(ParticipationID SERIAL PRIMARY KEY,
 ParticipantID INT NOT NULL,
 BookingID INT NOT NULL,
 StudentID CHAR(20) NULL,
 CHECK (StudentID IS NULL OR is_valid_student_id(StudentID)),
 FOREIGN KEY (ParticipantID) REFERENCES Participants,
 FOREIGN KEY (BookingID) REFERENCES ConferenceBookings);
```

3.15 DayParticipations

Tabela, do której wpisywane są uczestnictwa w danym dniu konferencji. Jeden wpis zawiera odwołania do ConferenceParticipations (ParticipationID) i DayBookings (DayBookingID).

```sql
CREATE TABLE IF NOT EXISTS DayParticipations
(ParticipationID INT NOT NULL,
 DayBookingID INT NOT NULL,
 PRIMARY KEY (ParticipationID, DayBookingID),
 FOREIGN KEY (DayBookingID) REFERENCES DayBookings,
 FOREIGN KEY (ParticipationID) REFERENCES ConferenceParticipations);
```

3.16 WorkshopParticipations

Jest tabelą, która przechowuje dane na temat uczestnictw w warsztatach. Zawiera odwołania do DayParticipations (ParticipationID, DayBookingID) oraz WorkshopBookings (WorkshopID, DayBookingID).

```sql
CREATE TABLE IF NOT EXISTS WorkshopParticipations
(ParticipationID INT NOT NULL,
 DayBookingID INT NOT NULL,
 WorkshopID INT NOT NULL,
 PRIMARY KEY (ParticipationID, DayBookingID, WorkshopID),
 FOREIGN KEY (ParticipationID, DayBookingID) REFERENCES
  ↪ DayParticipations,
 FOREIGN KEY (WorkshopID , DayBookingID) REFERENCES WorkshopBookings);
```

3.17 PriceTresholds

Tabela reprezentująca różne progi cenowe uczestnictwa w konferencji. Znajduje się tu pole z datą do której obowiązuje konkretna cena za udział (Until) oraz pole świadczące o wysokości przyznanej zniżki (Discount). Posiada ona także odwołanie do tabeli Conferences (ConfID).

```
CREATE TABLE IF NOT EXISTS PriceTresholds
(ConfID INT NOT NULL,
 Until TIMESTAMP NOT NULL,
 Discount DECIMAL NOT NULL,
 CHECK (Discount >= 0),
 PRIMARY KEY (ConfID, Until),
 FOREIGN KEY (ConfID) REFERENCES Conference);
```

3.18 Payments

To tabela, w której zapisane są płatności klientów. Każda płatność jest przypisana do rezerwacji jakiejś konferencji poprzez odwołanie do ConferenceBookings (BookingID). Płatności są dokumentowane poprzez przechowywanie informacji o dacie jej dokonania (When) oraz wysokości zrealizowanej wpłaty (Amount).

```
CREATE TABLE IF NOT EXISTS Payments
(PaymentID SERIAL PRIMARY KEY,
 BookingID INT NOT NULL,
 PaymentTime TIMESTAMP NOT NULL,
 Amount DECIMAL NOT NULL,
 FOREIGN KEY (BookingID) REFERENCES ConferenceBookings);
```

4 Widoki

4.1 days_and_free_seats

Widok ten wyświetla podstawowe informacje o dniach konferencji, a także ilość wolnych miejsc oraz cenę bazową.

```sql
CREATE VIEW days_and_free_seats AS
SELECT ConfID, ConfDate, StartTime, EndTime, Address, RoomNo,
 ↪ cd.NoSeats,
       cd.NoSeats - SUM(COALESCE(db.NoSeats, 0)) AS FreeSeats, BasePrice
  FROM ConferenceDay cd LEFT JOIN DayBookings db USING (ConfID,
  ↪ ConfDate)
  GROUP BY ConfID, ConfDate, StartTime, EndTime,
           Address, RoomNo, cd.NoSeats, BasePrice;
```

4.2 detailed_conferences

Widok ten jest poszerzeniem poprzedniego widoku - 'days_and_free_seats'. Dostarcza o dodatkowe informacje: temat konferencji, nazwę miasta, nazwę państwa, w którym odbywa się konferencja.

```sql
CREATE VIEW detailed_conferences AS
SELECT ConfID, ConfTopic, StartDate, EndDate, PostalCode, CityName AS
 ↪ City,
       SUM(cd.FreeSeats) AS UntakenSeats,
       SUM(cd.NoSeats) AS TotalSeats,
       BasePrice
  FROM Conference c
    NATURAL JOIN PostalCodes pc
    NATURAL JOIN Cities
    NATURAL JOIN days_and_free_seats cd
  GROUP BY ConfID, ConfTopic, StartDate, EndDate,
             PostalCode, CityName, BasePrice
  ORDER BY StartDate, ConfTopic;
```

4.3 conferences_history

Widok pokazuje informacje o odbytych konferencjach.

```sql
CREATE VIEW conferences_history AS
SELECT * FROM detailed_conferences
  WHERE NOW() > EndDate + INTERVAL '1 day'
  ORDER BY StartDate DESC, ConfTopic;
```

4.4 present_conferences

Widok pokazuje informacje o trwających konferencjach.

```
CREATE VIEW present_conferences AS
SELECT * FROM detailed_conferences
  WHERE NOW() < EndDate + INTERVAL '1 day' AND NOW() > StartDate
ORDER BY StartDate, ConfTopic;
```

4.5 detailed_company_customers

Widok wyświetla szczegółowe informacje o klientach firmowych.

```
CREATE VIEW detailed_company_customers AS
SELECT CustomerID, CompanyName AS CustomerName, CustomerType,
       Address, PostalCode, CityName AS City, CountryName AS Country,
       Phone || COALESCE(' ' || Fax, ' ') AS PhoneFax,
       Mail, Login
  FROM Customers
    NATURAL LEFT JOIN CompanyCustomers
    NATURAL JOIN PostalCodes
    NATURAL JOIN Cities
    NATURAL JOIN Countries
  WHERE CustomerType = 'Company'
  ORDER BY CompanyName;
```

4.6 detailed_individual_customers

Widok wyświetla szczegółowe informacje o klientach indywidualnych.

```
CREATE VIEW detailed_individual_customers AS
SELECT CustomerID, LastName || ' ' || FirstName AS CustomerName,
 ↪  CustomerType,
       Address, PostalCode, CityName AS City, CountryName AS Country,
       Phone AS PhoneFax,
       Mail, Login
  FROM Customers
    NATURAL LEFT JOIN IndividualCustomers
    NATURAL JOIN PostalCodes
    NATURAL JOIN Cities
    NATURAL JOIN Countries
  WHERE CustomerType = 'person'
  ORDER BY LastName || ' ' || FirstName;
```

4.7 detailed_customers

Widok zwraca szczegółowe informacje o wszystkich klientach: firmowych i indywidualnych.

```sql
CREATE VIEW detailed_customers AS
SELECT * FROM detailed_company_customers
UNION
SELECT * FROM detailed_individual_customers;
```

4.8 detailed_workshop_bookings

Widok ukazujący informacje o rezerwacjach warsztatów.

```sql
CREATE VIEW detailed_workshop_bookings AS
SELECT DayBookingID, WorkshopID, wb.NoSeats, NoStudents,
       (wb.NoSeats - NoStudents * StudentDiscount) AS WorkshopPrice
  FROM WorkshopBookings wb
    JOIN Workshop w USING (WorkshopID)
    JOIN ConferenceDay USING (ConfID, ConfDate)
    NATURAL JOIN Conference
  GROUP BY DayBookingID, WorkshopID, wb.NoSeats,
           w.BasePrice, wb.NoSeats, StudentDiscount
  ORDER BY DayBookingID DESC, WorkshopID DESC;
```

4.9 detailed_day_bookings

Widok wyświetla informacje o rezerwacjach dokonanych na poszczególne dni.

```sql
CREATE VIEW detailed_day_bookings AS
SELECT DayBookingID, BookingID, ConfID, ConfDate, db.NoSeats,
 ↪  db.NoStudents,
       (db.NoSeats - db.NoStudents * StudentDiscount) * BasePrice AS
        ↪  ConferenceDayPrice,
       SUM(COALESCE(WorkshopPrice, 0)) AS WorkshopsPrice
  FROM DayBookings db
    LEFT JOIN detailed_workshop_bookings USING (DayBookingID)
    JOIN ConferenceDay USING (ConfID, ConfDate)
    NATURAL JOIN Conference
  GROUP BY DayBookingID, BookingID, ConfID, ConfDate,
           db.NoSeats, db.NoStudents, StudentDiscount, BasePrice
  ORDER BY ConfDate DESC, ConfID DESC;
```

4.10 booking_total_payments

Widok sumy wszystkich wpłat dokonanych dla danej rezerwacji.

```
CREATE VIEW booking_total_payments AS
SELECT BookingID, SUM(COALESCE(Amount, 0)) AS AmountPaid
  FROM ConferenceBookings
    NATURAL LEFT JOIN Payments
  GROUP BY BookingID
  ORDER BY BookingID DESC;
```

4.11 detailed_bookings

Widok wyświetla szczegółowe informacje o rezerwacjach.

```
CREATE VIEW detailed_bookings AS
SELECT CustomerID, CustomerName, Address,
       PostalCode, City, Country, PhoneFax, Mail,
       BookingID, ConfID, ConfTopic, BookingTime,
       SUM(WorkshopsPrice) + SUM(ConferenceDayPrice) *
         (1 - COALESCE (
           (SELECT Discount
              FROM PriceTresholds pt
              WHERE pt.ConfID = cb.ConfID AND
                    BookingTime < Until + INTERVAL '1 day'
              ORDER BY Until DESC
              LIMIT 1), 0)) AS TotalPrice,
       SUM(WorkshopsPrice) + SUM(ConferenceDayPrice) AS
        ↪ TotalPriceNoDiscount,
       AmountPaid
  FROM ConferenceBookings cb
    NATURAL JOIN detailed_customers
    NATURAL JOIN booking_total_payments
    NATURAL LEFT JOIN detailed_day_bookings
    JOIN Conference USING (ConfID)
  GROUP BY CustomerID, CustomerName, Address, PostalCode,
             City, Country, PhoneFax, Mail, BookingID,
           ConfID, ConfTopic, BookingTime, AmountPaid
  ORDER BY BookingTime DESC, BookingID DESC;
```

4.12 overpaid_bookings

Widok wyświetla klientów, którzy zapłacili więcej niż wynosi faktyczny koszt uczestnictwa.

```sql
CREATE VIEW overpaid_bookings AS
SELECT CustomerID, CustomerName, Address,
       PostalCode, City, Country, PhoneFax, Mail,
       BookingID, ConfID, ConfTopic, BookingTime,
       TotalPrice, TotalPriceNoDiscount, AmountPaid,
       AmountPaid - TotalPrice AS AmountToReturn
  FROM detailed_bookings
  WHERE AmountPaid > TotalPrice
  ORDER BY BookingTime;
```

4.13 unpaid_bookings

Widok wyświetla klientów, którzy nie zapłacili jeszcze pełnej kwoty za dokonane rezerwacje.

```sql
CREATE VIEW unpaid_bookings AS
SELECT CustomerID, CustomerName, Address,
       PostalCode, City, Country, PhoneFax, Mail,
       BookingID, ConfID, ConfTopic, BookingTime,
       TotalPrice, TotalPriceNoDiscount, AmountPaid,
       TotalPrice - AmountPaid AS AmountToBePaid
  FROM detailed_bookings
  WHERE AmountPaid < TotalPrice
  ORDER BY BookingTime;
```

4.14 customers_activness

Widok udostępnia informacje klientach oraz o ich aktywności (liczbie dokonanych rezerwacji).

```sql
CREATE VIEW customers_activness AS
SELECT CustomerID, CustomerName, CustomerType,
       Address, PostalCode, City, Country, PhoneFax,
       Mail, Login, COUNT(BookingID) AS BookingsMade
  FROM detailed_customers
    NATURAL LEFT JOIN ConferenceBookings
  GROUP BY CustomerID, CustomerName, CustomerType,
           Address, PostalCode, City, Country, PhoneFax,
                  Mail, Login;
```

4.15 conference_popularity

Widok wyświetla najpopularniejsze konferencje oraz informacje o nich.

```sql
CREATE VIEW conference_popularity AS
SELECT dc.confid,
       dc.conftopic,
       dc.startdate || ' -- ' || dc.enddate AS conftime,
       dc.city,
       COUNT(StudentID) AS students,
       COUNT(conferenceparticipations.participantid) AS all_participants
  FROM detailed_conferences dc
    NATURAL LEFT JOIN conferencebookings
    NATURAL LEFT JOIN conferenceparticipations
  GROUP BY confid, conftopic, startdate, enddate, city
  ORDER BY all_participants DESC;
```

4.16 conference_popularity_among_students

Widok wyświetla konferencje, w których uczestniczyła największa liczba studentów.

```sql
CREATE VIEW conference_popularity_among_students AS
SELECT * FROM conference_popularity
  ORDER BY students DESC;
```

4.17 financial_stats

Widok pokazuje ilość zarobionych pieniędzy przez firmę organizującą konferencje z podziałem na lata i miesiące.

```sql
CREATE VIEW financial_stats AS
SELECT EXTRACT(YEAR FROM payments.paymenttime) AS year,
       EXTRACT(MONTH FROM payments.paymenttime) AS month,
       SUM(payments.amount) AS money_earned
FROM payments
GROUP BY ROLLUP(year, month)
ORDER BY year, month;
```

4.18 best_years

Widok pokazuje lata, w których firma organizująca konferencje zarobiła najwięcej.

```sql
CREATE VIEW best_years AS
SELECT EXTRACT(YEAR FROM payments.paymenttime) AS year,
       SUM(payments.amount) AS money_earned
FROM payments
GROUP BY year
ORDER BY money_earned DESC;
```

4.19 workshop_popularity

Widok wyświetla najpopularniejsze warsztaty, w których uczestniczyła największa liczba uczestników.

```sql
CREATE VIEW workshop_popularity AS
SELECT workshop.workshopid,
       workshop.workshoptopic,
       workshop.confdate AS date,
       city,
       workshop.address,
       COUNT(studentid) AS students,
       COUNT(ParticipationID) AS all_participants
  FROM workshop
    NATURAL JOIN detailed_conferences
    LEFT JOIN workshopbookings USING (WorkshopID)
    NATURAL LEFT JOIN workshopparticipations
    NATURAL LEFT JOIN dayparticipations
    NATURAL LEFT JOIN conferenceparticipations
  GROUP BY workshop.workshopid, city
  ORDER BY all_participants DESC;
```

4.20 workshop_popularity_among_students

Widok wyświetla warsztaty, w których uczestniczyła największa liczba studentów.

```sql
CREATE VIEW workshop_popularity_among_students AS
SELECT * FROM workshop_popularity
  ORDER BY students DESC
```

4.21 unfilled_workshop_bookings

Pokazuje informacje o rezerwacjach miejsc na warsztaty, na które nie została uzupełniona lista uczestników.

```sql
CREATE VIEW unfilled_workshop_bookings AS
SELECT DayBookingID, WorkshopID, wb.NoSeats - COUNT(ParticipationID) AS UnknownParticipants
  FROM WorkshopBookings wb
    NATURAL LEFT JOIN WorkshopParticipations
  GROUP BY DayBookingID, WorkshopID, wb.NoSeats
  HAVING COUNT(ParticipationID) < wb.NoSeats;
```

4.22 unfilled_day_bookings

Pokazuje informacje o rezerwacjach miejsc na konferencje, na które nie została uzupełniona lista uczestników.

```sql
CREATE VIEW unfilled_day_bookings AS
WITH unfilled_from_workshops AS
       (SELECT DayBookingID, SUM(UnknownParticipants) AS UnknownParticipants
          FROM unfilled_workshop_bookings
          GROUP BY DayBookingID)
SELECT BookingID, DayBookingID,
       db.NoSeats - COUNT(ParticipationID) AS UnknownConferenceParticipants,
       COALESCE(ufw.UnknownParticipants, 0) AS UnknownWorkshopsParticipants
  FROM DayBookings db
    NATURAL LEFT JOIN unfilled_from_workshops ufw
    NATURAL LEFT JOIN DayParticipations dp
  GROUP BY DayBookingID, ufw.UnknownParticipants
  HAVING COUNT(ParticipationID) < db.NoSeats OR
         ufw.UnknownParticipants IS NOT NULL;
```

4.23 unfilled_bookings

Pokazuje informacje o rezerwacjach na konferencje z nieuzupełnionymi listami uczestników oraz klientach, którzy je złożyli.

```sql
CREATE VIEW unfilled_bookings AS
SELECT CustomerID, CustomerName, Address,
       PostalCode, City, Country, PhoneFax, Mail,
       BookingID, ConfID, ConfTopic, BookingTime,
       StartDate - NOW() AS TimeLeft,
       SUM(UnknownConferenceParticipants) AS
        ↪  UnknownConferencesParticipants,
       SUM(UnknownWorkshopsParticipants) AS UnknownWorkshopsParticipants
  FROM ConferenceBookings cb
    NATURAL JOIN unfilled_day_bookings
    NATURAL JOIN detailed_customers
    JOIN Conference USING (ConfID)
  GROUP BY CustomerID, CustomerName, Address, PostalCode,
             City, Country, PhoneFax, Mail, BookingID,
           ConfID, ConfTopic, BookingTime, StartDate
  ORDER BY BookingTime, BookingID;
```

4.24 bookings_to_fill

Pokazuje informacje o klientach, z którymi należy skontaktować się w sprawie uzupełnienia listy zgłoszonych uczestników.

```sql
CREATE VIEW bookings_to_fill AS
SELECT * FROM unfilled_bookings
  WHERE TimeLeft < INTERVAL '1 week';
```

5 Funkcje

5.1 add_participant

Funkcja dodająca nowego uczestnika do bazy.

```sql
CREATE OR REPLACE FUNCTION add_participant(lastname varchar, firstname
↪ varchar, mail varchar)
RETURNS VOID AS $$
BEGIN
        INSERT INTO participants values(default, lastname, firstname,
        ↪ mail);
END;
$$ language plpgsql;
```

5.2 add_conference

Funkcja dodająca nową konferencję do bazy. Użytkownik podaje kod pocztowy lokalizacji, w której odbywa się konferencja, a przy wykorzystaniu funkcji get_id_from_postalcode(postalcode) do tabeli zostaje wpisane ID kodu pocztowego.

```sql
CREATE OR REPLACE FUNCTION add_conference(ctopic varchar, sdate date,
↪ edate date, postal varchar, sdiscount decimal)
RETURNS VOID AS $$
BEGIN
        INSERT INTO Conference VALUES (default, ctopic, sdate, edate,
        ↪ get_id_from_postalcode(postal), sdiscount);
END;
$$ LANGUAGE plpgsql;
```

5.3 get_id_from_postalcode

Funkcja zwracająca ID kodu pocztowego o podanej nazwie.

```sql
CREATE OR REPLACE FUNCTION get_id_from_postalcode(postal varchar)
RETURNS INT AS $idpc$
DECLARE idpc int;
BEGIN
        SELECT postalcodeid INTO idpc FROM postalcodes WHERE postalcode
        ↪ = postal;
        RETURN idpc;
END;
$idpc$ LANGUAGE plpgsql;
```

5.4 add_company_customer

Funkcja dodająca do bazy klientów firmowych.

```sql
CREATE OR REPLACE FUNCTION add_company_customer(
caddress varchar,
cpostalcode varchar,
cphone varchar,
clogin varchar,
cpassword varchar,
cmail varchar,
ccompanyname varchar,
cfax varchar) RETURNS VOID AS $$
DECLARE
        id int;
BEGIN
        INSERT INTO customers VALUES (default, 'company', caddress,
         ↪ get_id_from_postalcode(cpostalcode), cphone, clogin,
         ↪ cpassword, cmail)
        RETURNING customerid INTO id;
        INSERT INTO companycustomers VALUES (id, ccompanyname, cfax);
END;
$$ language plpgsql
```

5.5 add_individual_customer

Funkcja dodająca klientów indywidulanych.

```
CREATE OR REPLACE FUNCTION add_individual_customer(
caddress varchar,
cpostalcode varchar,
cphone varchar,
clogin varchar,
cpassword varchar,
cmail varchar,
clastname varchar,
cfirstname varchar
) RETURNS VOID AS $$
DECLARE
        id int;
BEGIN
        INSERT INTO customers VALUES (default, 'person', caddress,
        ↪ get_id_from_postalcode(cpostalcode), cphone, clogin,
        ↪ cpassword, cmail)
        RETURNING customerid INTO id;
        INSERT INTO individualcustomers VALUES (id, clastname,
        ↪ cfirstname);
END;
$$ language plpgsql
```

5.6 add_payment

Funkcja dodająca nową płatność do bazy.

```
CREATE OR REPLACE FUNCTION add_payment(
pbookingid INT,
ptime date,
pamount decimal
) RETURNS VOID AS $$
BEGIN
        INSERT INTO payments VALUES (default, pbookingid, ptime,
        ↪ pamount);
END;
$$ language plpgsql
```

5.7 add_conference_day

Funkcja dodająca nowy dzień konferencji do bazy.

```sql
CREATE OR REPLACE FUNCTION add_conference_day(
confid int,
confdate date,
daytopic varchar,
starttime time,
endtime time,
address varchar,
roomno varchar,
noseats int,
baseprice decimal
) RETURNS VOID AS $$
BEGIN
        INSERT INTO ConferenceDay VALUES (confid, confdate, daytopic,
         ↪  starttime, endtime, address, roomno, noseats, baseprice);
END;
$$ language plpgsql
```

5.8 add_workshop

```sql
CREATE OR REPLACE FUNCTION add_workshop(
confid int,
confdate date,
starttime time,
endtime time,
address varchar,
roomno varchar,
noseats int,
baseprice decimal
) RETURNS VOID AS $$
BEGIN
        INSERT INTO Workshop VALUES (default, confid, confdate,
         ↪  starttime, endtime, address, roomno, noseats, baseprice);
END;
$$ language plpgsql
```

5.9 add_pricetreshold

Funkcja dodająca nowy próg cenowy do bazy danych.

```
CREATE OR REPLACE FUNCTION add_pricetreshold(
confid int,
until time,
discout decimal
) RETURNS VOID AS $$
BEGIN
        INSERT INTO PriceTresholds VALUES (confid, until, discount);
END;
$$ language plpgsql
```

5.10 add_conference_booking

Funkcja dodaje nową rezerwację konferencji do bazy.

```
CREATE OR REPLACE FUNCTION add_conference_booking(
confid int,
until time,
customerid int,
bookingtime date
) RETURNS VOID AS $$
BEGIN
        INSERT INTO ConferenceBookings VALUES (default, until,
         ↪  customerid, bookingtime);
END;
$$ language plpgsql
```

5.11 add_day_booking

Funkcja ta dodaje rezerwacje na dany dzień koferencji.

```
CREATE OR REPLACE FUNCTION add_day_booking(
bookingid int,
confid int,
confdate time,
noseats int,
nostudents int
) RETURNS VOID AS $$
BEGIN
        INSERT INTO DayBookings VALUES (default, bookingid, confid,
         ↪  confdate, noseats, nostudents);
END;
$$ language plpgsql
```

5.12 add_workshop_booking

Dodaje rezerwacje na dany warsztat do bazy.

```
CREATE OR REPLACE FUNCTION add_workshop_booking(
daybookingid int,
workshopid int,
noseats int,
nostudents int
) RETURNS VOID AS $$
BEGIN
        INSERT INTO WorkshopBookings VALUES (daybookingid, workshopid,
         ↪  noseats, nostudents);
END;
$$ language plpgsql
```

5.13 add_postal_code

Funkcja dodaje nowy kod pocztowy do bazy.
Funkcja została poprawiona, aby nie dodawała miast i państw, jeśli już istnieją w bazie.

```
CREATE OR REPLACE FUNCTION add_postal_code(
postal_code varchar,
city_name varchar,
country_name varchar
) RETURNS VOID AS $$
DECLARE
        city int;
        country int;
BEGIN
        SELECT INTO country CountryID FROM Countries c WHERE CountryName
         ↪  = country_name;
        IF country IS NULL THEN
          INSERT INTO Countries VALUES (default, country_name) RETURNING
           ↪  countryid INTO country;
        END IF;
        SELECT INTO city CityID FROM Cities c WHERE CityName =
         ↪  city_name;
        IF city IS NULL THEN
          INSERT INTO Cities VALUES (default, city_name, country)
           ↪  RETURNING cityid INTO city;
        END IF;
        INSERT INTO PostalCodes VALUES (default, postalcode, city);
END;
$$ language plpgsql;
```

5.14 payment_booking_sum

Funkcja zwraca łączną kwotę wpłaconą na rzecz danej rezerwacji.
Niewymagany JOIN został usunięty i poprawiono literówkę.

```
CREATE OR REPLACE FUNCTION conference_day_booked_seats (cid INT, cdate
 ↪  DATE)
RETURNS INT AS $$
DECLARE
  booked_seats INT;
BEGIN
  booked_seats = (SELECT SUM(noseats)
                    FROM DayBookings db
                    WHERE db.confdate = cdate AND db.confid = cid);
  RETURN COALESCE(booked_seats, 0);
END;
$$ LANGUAGE plpgsql;
```

5.15 conference_day_booked_seats

Funkcja zwraca ilość zarezerwowanych miejsc dla danego dnia konferencji.

```
CREATE OR REPLACE FUNCTION conference_day_booked_seats (cid INT, cdate
 ↪  DATE)
RETURNS INT AS $$
DECLARE
        booked_seats INT;
BEGIN
        booked_seats = (SELECT SUM(noseats) FROM DayBookings db
                                        JOIN ConferenceDay cd ON
                                         ↪  db.confid = cd.confid AND
                                         ↪  db.confdate = cd.confdate
                                    WHERE db.confdate = cdate AND db.cid
                                     ↪  = db.confid);
        RETURN COALESCE(booked_seats, 0);
END;
$$ LANGUAGE plpgsql;
```

5.16 workshop_booked_seats

Funkcja zwracająca ilość zarezerwowanych miejsc na dany warsztat.
JOIN został poprawiony.

```
CREATE OR REPLACE FUNCTION workshop_booked_seats (wid INT)
RETURNS INT AS $$
DECLARE
  booked_seats INT;
BEGIN
  booked_seats = (SELECT SUM(noseats)
                    FROM WorkshopBookings
                    WHERE workshopid = wid);
RETURN COALESCE(booked_seats, 0);
END;
$$ LANGUAGE plpgsql;


\subsection{workshop\_participants\_list}
Funkcja zwraca tabele zawierającą dane osobe uczestników oraz adres
 ↪  mailowy dla wskazanego warsztatu.//
\textbf{Poprawiono JOIN'a.}
\begin{mysqlcode}
CREATE OR REPLACE FUNCTION workshop_participants_list (wid INT)
RETURNS TABLE (
participant_id INT,
last_name VARCHAR,
first_name VARCHAR,
email VARCHAR
) AS $$
BEGIN
RETURN QUERY
  SELECT Participants.ParticipantID,
         LastName, FirstName, Mail
    FROM Participants
      NATURAL JOIN ConferenceParticipations
      NATURAL JOIN WorkshopParticipations
      NATURAL JOIN WorkshopBookings
    WHERE WorkshopID = wid;
END;
$$ LANGUAGE plpgsql;
```

5.17 find_codes_for_city

Dla argumentu będącego nazwą miasta, jej częścią lub wrażeniem regularnym zwraca pasujące miasta razem z kodami pocztowymi.

```sql
CREATE OR REPLACE FUNCTION find_codes_for_city(city varchar) RETURNS
↪ SETOF RECORD AS $$
  SELECT CityName, PostalCodeID, PostalCode
    FROM Cities NATURAL LEFT JOIN PostalCodes
    WHERE SUBSTRING(CityName FROM city) IS NOT NULL;
$$ LANGUAGE SQL;
```

5.18 invalidate_late_unpaid_bookings

Dla każdej mającej już ponad tydzień nieopłaconej rezerwacji usuwa z bazy informację o liczbie zarezerwowanych miejsc i tym, komu były przypisane.

```sql
CREATE OR REPLACE FUNCTION invalidate_late_unpaid_bookings() RETURNS
↪ VOID AS $$
  DELETE FROM DayBookings
    WHERE BookingID IN
      (SELECT BookingID
            FROM unpaid_bookings
              WHERE NOW() > BookingTime + INTERVAL '1 week');
$$ LANGUAGE SQL;
```

5.19 remove_empty_bookings

Usuwa informacje o rezerwacjach na 0 miejsc nie mających żadnych wpłat.

```sql
CREATE OR REPLACE FUNCTION remove_empty_bookings() RETURNS VOID AS $$
  DELETE FROM ConferenceBookings
    WHERE BookingID IN
      (SELECT BookingID
         FROM ConferenceBookings
           NATURAL LEFT JOIN DayBookings
           NATURAL LEFT JOIN Payments
         WHERE DayBookingID IS NULL AND PaymentID IS NULL);
$$ LANGUAGE SQL;
```

5.20 add_day_participant

Dodaje uczestnika do danej listy zarezerwowanych miejsc na konferencję, zwraca id uczestnictwa, także w przypadku, gdy taka rezerwacja już istniała.

```
CREATE OR REPLACE FUNCTION add_day_participant(first_name varchar, last_name varchar,
                                               mailstr varchar, day_booking_id INT,
                                               out participation_id INT) AS $$
DECLARE
  participant_id INT;
BEGIN
  SELECT ParticipantID INTO participant_id
    FROM Participants
    WHERE LastName = last_name AND
          FirstName = first_name AND
          Mail = mailstr;

  IF part_id IS NULL THEN
    EXECUTE 'INSERT INTO Participants
               VALUES(DEFAULT, $1, $2, $3)
               RETURNIN ParticipantID'
      INTO STRICT participant_id
      USING last_name, first_name, mailstr;
  END IF;

  SELECT ParticipationID INTO participation_id
    FROM ConferenceParticipations
      NATURAL JOIN ConferenceBookings
      NATURAL JOIN DayBookings
    WHERE DayBookingID = day_id;

  IF participation_id IS NULL THEN
    EXECUTE 'INSERT INTO ConferenceParticipations
               VALUES(DEFAULT, $1, (SELECT BookingID FROM DayBookings
                                      WHERE DayBookingID = $2),
                      NULL) RETURNING ParticipationID'
      INTO STRICT participation_id
      USING participant_id, day_booking_id;
  END IF;

  IF NOT EXISTS (SELECT ParticipationID
                   FROM DayParticipations
                   WHERE DayBookingID = day_booking_id AND
                         ParticipationID = participation_id) THEN
    EXECUTE 'INSERT INTO DayParticipations
               VALUES($1, $2)'
      USING participation_id, day_booking_id;
  END IF;

END;
$$ LANGUAGE plpgsql;
```

5.21 add_workshop_participant

Dodaje uczestnika do danej listy zarezerwowanych miejsc na warsztat.

```
CREATE OR REPLACE FUNCTION add_workshop_participant(first_name varchar,
 ↪  last_name varchar,
                                                      mailstr varchar,
                                                       ↪  day_booking_id
                                                       ↪  INT,
                                                    workshop_id INT,
                                                    out participation_id
                                                     ↪  INT) AS $$
BEGIN
  participation_id :=
    add_day_participant(first_name, last_name, mailstr, day_booking_id);

  INSERT INTO WorkshopParticipations
    VALUES(participation_id, day_booking_id, workshop_id);
END;
$$ LANGUAGE plpgsql;
```

5.22 id_for_polish_city

Dla argumentu będącego nazwą polskiego miasta zwraca jego id dodając je do bazy, jeśli wcześniej nie było tam umieszczone.

```sql
CREATE OR REPLACE FUNCTION id_for_polish_city(city VARCHAR, out id INT) AS $$
DECLARE
poland INT;
BEGIN
SELECT INTO id CityID
FROM Cities
WHERE CityName = city;
IF id IS NULL THEN
SELECT INTO STRICT poland CountryID
FROM Countries
WHERE CountryName = 'Polska';
EXECUTE 'INSERT INTO Cities VALUES(DEFAULT, $1, $2) RETURNING CityID'
INTO STRICT id
USING city, poland;
END IF;
END;
$$ LANGUAGE plpgsql;
```

5.23 is_valid_mail

```sql
CREATE OR REPLACE FUNCTION is_valid_mail(mailstring varchar) RETURNS boolean AS
$$
SELECT mailstring SIMILAR TO '_+@_+._+';
$$ LANGUAGE SQL;
```

5.24 is_valid_phone_or_fax

```sql
CREATE OR REPLACE FUNCTION is_valid_phone_or_fax(p_or_f char) RETURNS boolean AS
$$
  SELECT p_or_f IS NULL OR p_or_f SIMILAR TO '\+?[[:digit:]]{3,15}';
$$ LANGUAGE SQL;
```

5.25 is_valid_login

```sql
CREATE OR REPLACE FUNCTION is_valid_login(login char) RETURNS boolean AS
$$
  SELECT login SIMILAR TO '[[:alnum:]][._.]-]+';
$$ LANGUAGE SQL;
```

5.26 is_valid_polish_zip

```sql
CREATE OR REPLACE FUNCTION is_valid_polish_zip(zip char) RETURNS boolean
  ↪ AS
$$
  SELECT zip SIMILAR TO '[[:digit:]]{2}-[[:digit:]]{3}';
$$ LANGUAGE SQL;
```

5.27 is_valid_name

```sql
CREATE OR REPLACE FUNCTION is_valid_name(namestring varchar) RETURNS
  ↪ BOOLEAN AS
$$
SELECT namestring SIMILAR TO '[[:alpha:]]+([''-][[:alpha:]]+)*';
$$ LANGUAGE SQL;
```

5.28 is_valid_student_id

```sql
CREATE OR REPLACE FUNCTION is_valid_student_id(id char) RETURNS BOOLEAN
  ↪ AS
$$
  SELECT id SIMILAR TO '[[:alnum:]]+';
$$ LANGUAGE SQL;
```

5.29 get_customer_type

Funkcja pomocnicza, zwraca typ klienta.

```sql
CREATE OR REPLACE FUNCTION get_customer_type(id INT) RETURNS VARCHAR AS
  ↪ $$
  SELECT CustomerType FROM Customers c WHERE id = c.CustomerID;
$$ LANGUAGE SQL;
```

5.30 does_customerid_appear_in

Funkcja pomocnicza, sprawdza czy w podanej tabeli znajduje się CustomerID o podanej wartości.

```sql
CREATE OR REPLACE FUNCTION does_customerid_appear_in(id INT, queried_table VARCHAR, OUT res BOOLEAN) AS $$
  BEGIN
    EXECUTE FORMAT('SELECT EXISTS (
                    SELECT * FROM %s c
                             WHERE c.CustomerID = $1)',
                  queried_table)
      INTO STRICT res
      USING id;
  END;
$$ LANGUAGE plpgsql;
```

5.31 time_ranges_collide

Funkcja pomocnicza, sprawdza, czy dwa przedziały czasowe na siebie nachodzą

```sql
CREATE OR REPLACE FUNCTION time_ranges_collide(start1 TIME, end1 TIME, start2 TIME,
                                              end2 TIME, OUT res BOOLEAN) AS $$
  BEGIN
    res := end2 > start1 AND start2 < end1;
  END;
$$ LANGUAGE plpgsql;
```

6 Triggery

6.1 T_validate_postal_code

Jeśli mamy do czynienia z polskim kodem pocztowym, sprawdza jego poprawność

```sql
CREATE OR REPLACE FUNCTION validate_postal_code() RETURNS TRIGGER AS $$
  DECLARE
    country VARCHAR;
  BEGIN
    EXECUTE 'WITH id AS (
              SELECT CountryName FROM Countries, id
              WHERE Countries.CountryID = id.CountryID'
             SELECT CountryName FROM Countries
               WHERE CountryID = id'
      INTO STRICT country
      USING NEW.CityID;

    IF country = 'Polska' AND NOT is_valid_polish_zip(NEW.PostalCode)
    ↪ THEN
      RAISE EXCEPTION '"%" is not a valid polish zip code',
    ↪ NEW.PostalCode;
    END IF;
    RETURN NEW;
  END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER T_validate_postal_code BEFORE INSERT OR UPDATE ON
  ↪ PostalCodes
  FOR EACH ROW EXECUTE PROCEDURE validate_postal_code();
```

6.2 T_limit_day_places

Sprawdza, czy na dzień konferencji nie jest zapisanych więcej osób, niż przewidujemy.

```
CREATE OR REPLACE FUNCTION limit_day_places() RETURNS TRIGGER AS $$
  DECLARE
    free_places INT;
  BEGIN
    EXECUTE 'SELECT cd.NoSeats - SUM(COALESCE(db.NoSeats, 0))
                  FROM DayBookings db RIGHT JOIN ConferenceDay cd
                ON ((db.ConfID, db.ConfDate) = ($1, $2) AND
                    (cd.ConfID, cd.ConfDate) = ($1, $2))
              GROUP BY cd.NoSeats'
      INTO STRICT free_places
      USING NEW.ConfID, NEW.ConfDate;

    IF free_places < 0 THEN
      RAISE EXCEPTION 'Not enough free places on conference on %. Would
       ↪  need % more.',
        NEW.ConfDate, -free_places;
    END IF;
    RETURN NEW;
  END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER T_limit_day_places AFTER INSERT OR UPDATE ON DayBookings
  FOR EACH ROW EXECUTE PROCEDURE limit_day_places();

CREATE TRIGGER T_limit_day_places AFTER UPDATE ON ConferenceDay
  FOR EACH ROW EXECUTE PROCEDURE limit_day_places();
```

6.3 T_limit_workshop_places

Sprawdza, czy na warsztat nie jest zapisanych więcej osób niż przewidujemy

```
CREATE OR REPLACE FUNCTION limit_workshop_places() RETURNS TRIGGER AS $$
  DECLARE
    free_places INT;
    conf_date DATE;
  BEGIN
    EXECUTE 'SELECT w.NoSeats - SUM(COALESCE(wb.NoSeats, 0)), w.ConfDate
               FROM WorkshopBookings wb RIGHT JOIN Workshop w
                 ON (w.WorkshopID = $1 AND
                     wb.WorkshopID = $1)
               GROUP BY w.NoSeats, w.ConfDate'
      INTO STRICT free_places, conf_date
      USING NEW.WorkshopID;

    IF free_places < 0 THEN
      RAISE EXCEPTION 'Not enough free places on workshop % on %. Would
       ↪  need % more.',
        NEW.WorkshopID, conf_date, -free_places;
    END IF;
    RETURN NEW;
  END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER T_limit_workshop_places AFTER INSERT OR UPDATE ON
 ↪  WorkshopBookings
  FOR EACH ROW EXECUTE PROCEDURE limit_workshop_places();
CREATE TRIGGER T_limit_workshop_places AFTER UPDATE ON Workshop
  FOR EACH ROW EXECUTE PROCEDURE limit_workshop_places();
```

6.4 T_ensure_complete_customer_info

Sprawdza, czy klient, do którego ma być przypisana rezerwacja ma przypisane odpowiednie dane w IndividualCustomers albo CompanyCustomers.

```sql
CREATE OR REPLACE FUNCTION ensure_complete_customer_info() RETURNS
↪ TRIGGER AS $$
  DECLARE
    queried VARCHAR := 'IndividualCustomers';
    id INT := NEW.CustomerID;
  BEGIN
    IF TG_TABLE_NAME IN ('IndividualCustomers', 'CompanyCustomers') THEN
      id := OLD.CustomerID;
    END IF;

    IF get_customer_type(id) = 'Company' THEN
      queried := 'CompanyCustomers';
    END IF;

    IF does_customerid_appear_in(id, 'ConferenceBookings') AND
        NOT does_customerid_appear_in(id, queried) THEN
      RAISE EXCEPTION 'Attempt to make customer of id % have conference
       ↪ bookings '
        'assigned while not having information assigned in %.', id,
         ↪ queried;
    END IF;
    RETURN NEW;
  END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER T_ensure_complete_customer_info AFTER UPDATE OR INSERT ON
↪ ConferenceBookings
  FOR EACH ROW EXECUTE PROCEDURE ensure_complete_customer_info();

CREATE TRIGGER T_ensure_complete_customer_info AFTER DELETE OR UPDATE ON
↪ IndividualCustomers
  FOR EACH ROW EXECUTE PROCEDURE ensure_complete_customer_info();

CREATE TRIGGER T_ensure_complete_customer_info AFTER DELETE OR UPDATE ON
↪ CompanyCustomers
  FOR EACH ROW EXECUTE PROCEDURE ensure_complete_customer_info();
```

6.5 T_validate_price_treshold_dates

Sprawdza, czy daty obowiązywania zniżek na konferencję nie przekraczają daty jej rozpozęcia.

```
CREATE OR REPLACE FUNCTION validate_price_treshold_dates() RETURNS
↪  TRIGGER AS $$
  DECLARE
    conf_start DATE;
    exists_invalid_value BOOLEAN;
  BEGIN
    EXECUTE 'SELECT EXISTS (
               SELECT *
                 FROM Conference c NATURAL JOIN PriceTresholds pt
                 WHERE ConfID = $1 AND
                       c.StartDate <= pt.Until)'
      INTO STRICT exists_invalid_value
      USING NEW.ConfID;

    IF exists_invalid_value THEN
      RAISE EXCEPTION 'Attempt to make price treshold for conference % '
        'last longer than conference''s start date.',
        NEW.ConfID;
    END IF;
    RETURN NEW;
  END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER T_validate_price_treshold_dates AFTER INSERT OR UPDATE ON
↪  PriceTresholds
  FOR EACH ROW EXECUTE PROCEDURE validate_price_treshold_dates();

CREATE TRIGGER T_validate_price_treshold_dates AFTER INSERT OR UPDATE ON
↪  Conference
  FOR EACH ROW EXECUTE PROCEDURE validate_price_treshold_dates();
```

6.6 T_validate_booking_dates

Sprawdza, czy ktoś nie rezerwował konferencji już po jej rozpoczęciu.

```sql
CREATE OR REPLACE FUNCTION validate_booking_dates() RETURNS TRIGGER AS $$
  DECLARE
    conf_start DATE;
    exists_invalid_value BOOLEAN;
  BEGIN
    EXECUTE 'SELECT EXISTS (
               SELECT *
                        FROM Conference c NATURAL JOIN ConferenceBookings cb
                 WHERE ConfID = $1 AND
                       c.StartDate < cb.BookingTime)'
      INTO STRICT exists_invalid_value
      USING NEW.ConfID;

    IF exists_invalid_value THEN
      RAISE EXCEPTION 'Attempt to make booking for conference % '
        'newer than conference''s start date.',
        NEW.ConfID;
    END IF;
    RETURN NEW;
  END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER T_validate_booking_dates AFTER INSERT OR UPDATE ON ConferenceBookings
  FOR EACH ROW EXECUTE PROCEDURE validate_booking_dates();

CREATE TRIGGER T_validate_booking_dates AFTER INSERT OR UPDATE ON Conference
  FOR EACH ROW EXECUTE PROCEDURE validate_booking_dates();
```

6.7 T_ensure_unique_postal_codes

Sprawdza, czy kody pocztowe nie powtarzają się w obrębie kraju.

```sql
CREATE OR REPLACE FUNCTION ensure_unique_postal_codes() RETURNS TRIGGER AS $$
  DECLARE
    exists_invalid_value BOOLEAN;
  BEGIN
    EXECUTE 'SELECT EXISTS (
               WITH codes_with_countries AS (
                 SELECT PostalCode pc, PostalCodeID pcid, CountryID cid
                          FROM PostalCodes NATURAL JOIN Cities NATURAL JOIN Countries)
               SELECT *
                 FROM codes_with_countries c1 JOIN codes_with_countries c2
                   ON (c1.pcid = $1 AND c1.pcid <> c2.pcid AND
                       c1.pc = c2.pc AND c1.cid = c2.cid))'
      INTO STRICT exists_invalid_value
      USING NEW.PostalCodeID;

    IF exists_invalid_value THEN
      RAISE EXCEPTION 'Attempt to assing postal code % twice.',
        NEW.PostalCode;
    END IF;
    RETURN NEW;
  END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER T_ensure_unique_postal_codes AFTER INSERT OR UPDATE ON PostalCodes
  FOR EACH ROW EXECUTE PROCEDURE ensure_unique_postal_codes();
```

6.8 T_limit_workshop_booking_places

Sprawdza, czy do rezerwacji warsztatu nie jest przypisanych więcej osób, niż podane w rezerwacji.

```sql
CREATE OR REPLACE FUNCTION limit_workshop_booking_places() RETURNS TRIGGER AS $$
  DECLARE
    free_places INT;
  BEGIN
    EXECUTE 'SELECT wb.NoSeats - COUNT(wp.ParticipationID)
                   FROM WorkshopBookings wb LEFT JOIN WorkshopParticipations wp
                 ON ((wb.WorkshopID, wb.DayBookingID) = ($1, $2) AND
                     (wp.WorkshopID, wp.DayBookingID) = ($1, $2))
               GROUP BY wb.NoSeats'
      INTO STRICT free_places
      USING NEW.WorkshopID, NEW.DayBookingID;

    IF free_places < 0 THEN
      RAISE EXCEPTION 'Not enough booked places in day booking % for workshop %. Would need % more.',
        NEW.DayBookingID, NEW.WorkshopID, -free_places;
    END IF;
    RETURN NEW;
  END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER T_limit_workshop_booking_places AFTER INSERT OR UPDATE ON WorkshopParticipations
  FOR EACH ROW EXECUTE PROCEDURE limit_workshop_booking_places();

CREATE TRIGGER T_limit_workshop_booking_places AFTER UPDATE ON WorkshopBookings
  FOR EACH ROW EXECUTE PROCEDURE limit_workshop_booking_places();
```

6.9 T_limit_day_booking_places

Sprawdza, czy do rezerwacji konferencji w danym dniu nie jest przypisanych więcej osób, niż podane w rezerwacji.

```
CREATE OR REPLACE FUNCTION limit_day_booking_places() RETURNS TRIGGER AS
↪ $$
  DECLARE
    free_places INT;
  BEGIN
    EXECUTE 'SELECT db.NoSeats - COUNT(dp.ParticipationID)
                  FROM DayBookings db LEFT JOIN DayParticipations dp
                ON (db.DayBookingID = $1 AND
                    dp.DayBookingID = $1)
              GROUP BY db.NoSeats'
      INTO STRICT free_places
      USING NEW.DayBookingID;

    IF free_places < 0 THEN
      RAISE EXCEPTION 'Not enough booked places in day booking %. Would
      ↪ need % more.',
        NEW.DayBookingID, -free_places;
    END IF;
    RETURN NEW;
  END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER T_limit_day_booking_places AFTER INSERT OR UPDATE ON
↪ DayParticipations
  FOR EACH ROW EXECUTE PROCEDURE limit_day_booking_places();

CREATE TRIGGER T_limit_day_booking_places AFTER UPDATE ON DayBookings
  FOR EACH ROW EXECUTE PROCEDURE limit_day_booking_places();
```

6.10 T_ensure_valid_participations

Sprawdza, czy krotka opisująca uczestnictwo w dniu konferencji nie jest powiązana z uczestnictwem w konferencji i rezerwacją konferencji na danny dzień powiązanymi z różnymi rezerwacjami całej konferencji.

```
CREATE OR REPLACE FUNCTION ensure_valid_participations() RETURNS TRIGGER AS $$
  DECLARE
    is_invalid_participation BOOLEAN;
  BEGIN
    IF TG_TABLE_NAME = 'DayBookings' AND NEW.BookingID <> OLD.BookingID THEN
      EXECUTE 'EXISTS (SELECT * FROM DayParticipations
                         WHERE DayBookingID = $1)'
        INTO STRICT is_invalid_participation
        USING NEW.DayBookingID;
      IF is_invalid_participation THEN
        RAISE EXCEPTION 'Can''t link day booking % to different conference booking '
          'while there are participations linked to it.',
          NEW.DayBookingID;
      END IF;
    END IF;
    IF TG_TABLE_NAME = 'ConferenceParticipations' AND NEW.BookingID <> OLD.BookingID THEN
      EXECUTE 'EXISTS (SELECT * FROM DayParticipations
                         WHERE PaticipationID = $1)'
        INTO STRICT is_invalid_participation
        USING NEW.ParticipationID;
      IF is_invalid_participation THEN
        RAISE EXCEPTION 'Can''t link day conference participation % to different booking '
          'while there are day participations linked to it.',
          NEW.ParticipationID;
      END IF;
    END IF;
    IF TG_TABLE_NAME = 'DayParticipations' THEN
      EXECUTE 'EXISTS (SELECT *
                               FROM ConferenceBookings cb1 NATURAL JOIN DayBookings db JOIN DayParticipations dp
                           ON (db.DayBookingID = $1 AND dp.DayBookingID = $1)
                           JOIN ConferenceParticipations cp
```

```
                            ON (dp.ParticipationID = $2 AND
 ↪  cp.ParticipationID = $2)
                            JOIN ConferenceBookings cb2
                           ON (cp.BookingID = cb2.BookingID)
                         WHERE cb1.BookingID <> cb2.BookingID)'
        INTO STRICT is_invalid_participation
        USING NEW.DayBookingID, NEW.ParticipationID;

      IF is_invalid_participation THEN
        RAISE EXCEPTION 'Participation % and day booking % link to
         ↪  different conference bookings.',
          NEW.ParticipationID, NEW.DayBookingID;
      END IF;
    END IF;
    RETURN NEW;
  END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER T_ensure_valid_participations AFTER INSERT OR UPDATE ON
 ↪  DayParticipations
  FOR EACH ROW EXECUTE PROCEDURE ensure_valid_participations();

CREATE TRIGGER T_ensure_valid_participations AFTER UPDATE ON DayBookings
  FOR EACH ROW EXECUTE PROCEDURE ensure_valid_participations();

CREATE TRIGGER T_ensure_valid_participations AFTER UPDATE ON
 ↪  ConferenceParticipations
  FOR EACH ROW EXECUTE PROCEDURE ensure_valid_participations();
```

6.11 T_limit_non_students

Sprawdza, czy nie-studenci nie mają zarezerwowanych miejsc jako studenci.

```
CREATE OR REPLACE FUNCTION limit_non_students() RETURNS TRIGGER AS $$
  DECLARE
    lacking_booked_adult_places BOOLEAN;
  BEGIN
    IF TG_TABLE_NAME = 'ConferenceParticipations' AND NEW.StudentID IS
     ↪  NULL AND
                         OLD.StudentID IS NOT NULL THEN
      EXECUTE 'SELECT EXISTS (
                   SELECT db.DayBookingID
                     FROM DayParticipations dp1 JOIN DayBookings db
                       ON (dp1.ParticipationID = $1 AND dp1.DayBookingID =
 ↪  db.DayBookingID)
                       JOIN DayParticipations dp2
                       ON (dp2.DayBookingID = db.DayBookingID)
                       JOIN ConferenceParticipations cp
                       ON (dp2.ParticipationID = cp.ParticipationID)
                     GROUP BY db.DayBookingID
                     HAVING COUNT(cp.ParticipationID) -
 ↪  COUNT(cp.StudentID) >
                            db.NoSeats - db.NoStudents)'
        INTO STRICT lacking_booked_adult_places
        USING NEW.ParticipationID;
      IF lacking_booked_adult_places THEN
        RAISE EXCEPTION 'Too little non-student seats booked.';
      END IF;
    END IF;
    IF TG_TABLE_NAME IN ('DayParticipations', 'DayBookings') THEN
      EXECUTE 'SELECT EXISTS (
                   SELECT db.DayBookingID
                     FROM DayBookings db JOIN DayParticipations dp
                       ON (dp.DayBookingID = $1 AND db.DayBookingID = $1)
                       JOIN ConferenceParticipations cp
                       ON (dp.ParticipationID = cp.ParticipationID)
                     GROUP BY db.DayBookingID
                     HAVING COUNT(cp.ParticipationID) -
 ↪  COUNT(cp.StudentID) >
                            db.NoSeats - db.NoStudents)'
        INTO STRICT lacking_booked_adult_places
        USING NEW.DayBookingID;
      IF lacking_booked_adult_places THEN
```

```
        RAISE EXCEPTION 'Too little non-student seats booked in day
         ↪  booking %.',
        NEW.DayBookingID;
      END IF;
    END IF;
    RETURN NEW;
  END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER T_limit_non_students AFTER INSERT OR UPDATE ON
 ↪  DayParticipations
  FOR EACH ROW EXECUTE PROCEDURE limit_non_students();

CREATE TRIGGER T_limit_non_students AFTER UPDATE ON DayBookings
  FOR EACH ROW EXECUTE PROCEDURE limit_non_students();

CREATE TRIGGER T_limit_non_students AFTER UPDATE ON
 ↪  ConferenceParticipations
  FOR EACH ROW EXECUTE PROCEDURE limit_non_students();
```

6.12 T_forbid_colliding_workshop_participations

Sprawdza, czy ktoś nie jest wpisany na nachodzące na siebie warsztaty.

```
CREATE OR REPLACE FUNCTION forbid_colliding_workshop_participations()
 ↪  RETURNS TRIGGER AS $$
  DECLARE
    detected_colliding_participations BOOLEAN;
  BEGIN
    IF TG_TABLE_NAME = 'WorkshopParticipations' THEN
      EXECUTE 'SELECT EXISTS (
                 SELECT w1.WorkshopID
                   FROM Workshop w1 JOIN WorkshopParticipations wp
                     ON (w1.WorkshopID = $1 AND
                         wp.WorkshopID <> $1 AND
                         (wp.ParticipationID, wp.DayBookingID) = ($2,
 ↪  $3))
                     JOIN Workshop w2
                     ON (w2.WorkshopID = wp.WorkshopID AND
                         time_ranges_collide(w1.StartTime, w1.EndTime,
                                             w2.StartTime >
 ↪  w2.EndTime)))'
        INTO STRICT detected_colliding_participations
        USING NEW.WorkshopID, NEW.ParticipationID, NEW.DayBookingID;
    END IF;
    IF TG_TABLE_NAME = 'Workshop' THEN
      EXECUTE 'SELECT EXISTS (
                 SELECT w1.WorkshopID
                   FROM  WorkshopParticipations wp1 JOIN WorkshopParticipations wp2
                     ON (wp1.WorkshopID = $1 AND
                         wp2.WorkshopID <> $1 AND
                         (wp1.ParticipationID, wp1.DayBookingID) = (wp2.ParticipationID, wp2.DayBookingID))
                     JOIN Workshop w
                     ON (w.WorkshopID = wp2.WorkshopID AND
                         time_ranges_collide($2, $3, w2.StartTime, w2.EndTime)))'
        INTO STRICT detected_colliding_participations
        USING NEW.WorkshopID, NEW.StartTime, NEW.EndTime;
    END IF;
    IF detected_colliding_participations THEN
      RAISE EXCEPTION 'Can''t have a participant assigned to two colliding workshops.';
    END IF;
    RETURN NEW;
  END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER T_forbid_colliding_workshop_participations AFTER INSERT OR UPDATE ON WorkshopParticipations
  FOR EACH ROW EXECUTE PROCEDURE forbid_colliding_workshop_participations();

CREATE TRIGGER T_forbid_colliding_workshop_participations AFTER UPDATE ON Workshop
  FOR EACH ROW EXECUTE PROCEDURE forbid_colliding_workshop_participations();
```

6.13 T_forbid_conference_collisions

Sprawdza, czy warsztat nie nachodzi na konferencję.

```sql
CREATE OR REPLACE FUNCTION forbid_conference_collisions() RETURNS TRIGGER AS $$
  DECLARE
    detected_collision BOOLEAN := FALSE;
  BEGIN
    IF TG_TABLE_NAME = 'Workshop' THEN
      EXECUTE 'SELECT EXISTS (
                 SELECT cd.ConfId
                   FROM ConferenceDay
                   WHERE (cd.ConfID, cd.ConfDate) = ($1, $2) AND
                     time_ranges_collide($3, $4, cd.StartTime, cd.EndTime))'
        INTO STRICT detected_collision
        USING NEW.ConfID, NEW.ConfDate, NEW.StartTime, NEW.EndTime;
    END IF;
    IF TG_TABLE_NAME = 'ConferenceDay' THEN
      EXECUTE 'SELECT EXISTS (
                 SELECT w.WorkshopID
                   FROM  Workshop w
                   WHERE (w.ConfID, w.ConfDate) = ($1, $2) AND
                     time_ranges_collide($3, $4, w.StartTime, w.EndTime))'
        INTO STRICT detected_collision
        USING NEW.ConfID, NEW.ConfDate, NEW.StartTime, NEW.EndTime;
    END IF;
    IF detected_collision THEN
      RAISE EXCEPTION 'Attempt to make Conference % % collide with own workshop.',
        NEW.ConfID, NEW.ConfDate;
    END IF;
    RETURN NEW;
  END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER T_forbid_conference_collisions AFTER INSERT OR UPDATE ON Workshop
  FOR EACH ROW EXECUTE PROCEDURE forbid_conference_collisions();

CREATE TRIGGER T_forbid_conference_collisions AFTER UPDATE ON ConferenceDay
  FOR EACH ROW EXECUTE PROCEDURE forbid_conference_collisions();
```

6.14 T_ensure_valid_conference_date

Sprawdza, czy dzień konferencji mieści się w przedziale czasowym całej konferencji.

```sql
CREATE OR REPLACE FUNCTION ensure_valid_conference_date() RETURNS TRIGGER AS $$
  DECLARE
    detected_invalid_date BOOLEAN := FALSE;
  BEGIN
    IF TG_TABLE_NAME = 'ConferenceDay' THEN
      EXECUTE 'SELECT EXISTS (
                 SELECT c.ConfId
                   FROM Conference
                   WHERE c.ConfID = $1 AND
                     ($2 < c.StartDate OR $2 > c.EndDate))'
        INTO STRICT detected_invalid_date
        USING NEW.ConfID, NEW.ConfDate;
    END IF;
    IF TG_TABLE_NAME = 'Conference' THEN
      EXECUTE 'SELECT EXISTS (
                 SELECT cd.ConfID
                   FROM  ConferenceDay cd
                   WHERE cd.ConfID = $1 AND
                     (cd.ConfDate < $2 OR cd.ConfDate > $3))'
        INTO STRICT detected_invalid_date
        USING NEW.ConfID, NEW.StartDate, NEW.EndDate;
    END IF;
    IF detected_invalid_date THEN
      RAISE EXCEPTION 'Attempt to make a day of conference % outside it''s time range',
        NEW.ConfID;
    END IF;
    RETURN NEW;
  END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER T_ensure_valid_conference_date AFTER INSERT OR UPDATE ON ConferenceDay
  FOR EACH ROW EXECUTE PROCEDURE ensure_valid_conference_date();

CREATE TRIGGER T_ensure_valid_conference_date AFTER UPDATE ON Conference
  FOR EACH ROW EXECUTE PROCEDURE ensure_valid_conference_date();
```

7 Indeksy

7.1 conference_confid_index

Indeks utworzony na ID konferncji.

```sql
CREATE INDEX  conference_confid_index ON Conference (ConfID)
```

7.2 participantid_index

Indeks utworzony na ID uczestnika.

```sql
CREATE INDEX  participantid_index ON Participants (ParticipantID)
```

8 Role

8.1 Administrator

Osoba specjalizująca się w obsłudze systemów bazodanowych, biegle posługująca się językiem SQL. Posiada możliwość do ulepszania i rozbudowy bazy danych. Posiada on także dostęp do wszystkich funkcji oraz widoków.

8.2 Organizator warsztatów

Osoba ta jest odpowiedzialna za organizację oraz przebieg warsztatów. Posiada on uprawnienia do dodawania nowego cyklu warsztatów do bazy, jak również posiada dostęp do funkcji oraz widoków dotyczących warsztatów (np. listy uczestników, ilości wolnych miejsc na warsztat).

8.3 Uczestnik konferencji

Jest to osoba fizyczna uczestnicząca w konferencji. Posiada dostęp do widoków zbliżających się, odbytych, bądź właśnie trwających konferencji, warsztatów.

8.4 Klient

Klientem jest osoba indywidualna lub firma. Mianem klienta określamy firmę, bądź osobę fizyczną, która dokonała rezerwacji na dowolną konferencje. Klienci posiadają dostęp do widoków dokonanych rezerwacji oraz funkcji dodawania nowego uczestnika.

8.5 Zarząd firmy

Zarząd na czele z prezesem firmy posiada dostęp do statystyk finansowych oraz widoku najlepszych lat działalności firmy. Ponadto posiada on możliwość do tworzenia nowej konferencji, a także decyduje o progach cenowych i wysokości zniżek studenckich.

9 Generator

Do wygenerowania przykładowych danych do naszej bazy skorzystaliśmy z programu: **Datanamic Data Generator**. Przykładowy fragment wygenerowanego skryptu znajduje się poniżej:

```sql
INSERT INTO "public"."countries" ("countryid","countryname") VALUES
 ↪ (1,'Iraq');


INSERT INTO "public"."countries" ("countryid","countryname") VALUES
 ↪ (2,'El Salvador');


INSERT INTO "public"."countries" ("countryid","countryname") VALUES
 ↪ (3,'Monaco');


INSERT INTO "public"."countries" ("countryid","countryname") VALUES
 ↪ (4,'Cyprus');


INSERT INTO "public"."countries" ("countryid","countryname") VALUES
 ↪ (5,'Cook Islands');





INSERT INTO "public"."participants"
 ↪ ("participantid","lastname","firstname","mail") VALUES
 ↪ (1,'Crocetti','Barbara','wlgjjup@gaak.pl');


INSERT INTO "public"."participants"
 ↪ ("participantid","lastname","firstname","mail") VALUES
 ↪ (2,'Orcutt',NULL,'ghltyuf@lybh.pl');


INSERT INTO "public"."participants"
 ↪ ("participantid","lastname","firstname","mail") VALUES
 ↪ (3,'Langham         ','Pawel','lojdxbv@ryvi.pl');


```

```
INSERT INTO "public"."participants"
↪  ("participantid","lastname","firstname","mail") VALUES
↪  (4,'McDaniel',NULL,'ewzorig@tftx.com');


INSERT INTO "public"."participants"
↪  ("participantid","lastname","firstname","mail") VALUES
↪  (5,'Caffray','Rachael','krxunoa@firy.com');


INSERT INTO "public"."participants"
↪  ("participantid","lastname","firstname","mail") VALUES
↪  (6,'Deleo','Caitlin','aapoevs@ujys.pl');


INSERT INTO "public"."participants"
↪  ("participantid","lastname","firstname","mail") VALUES
↪  (7,'Julieze','Cath','alldbvh@uysx.com');


INSERT INTO "public"."participants"
↪  ("participantid","lastname","firstname","mail") VALUES
↪  (8,NULL,'Juana','znevhvm@deio.pl');


INSERT INTO "public"."participants"
↪  ("participantid","lastname","firstname","mail") VALUES
↪  (9,'Arnold','Lea','vyatzgo@gusf.pl');


INSERT INTO "public"."participants"
↪  ("participantid","lastname","firstname","mail") VALUES
↪  (10,'Brown','Frederik','pyqgdgq@zizh.com');





INSERT INTO "public"."cities" ("cityid","cityname","countryid") VALUES
↪  (1,'Nagpur',3);


INSERT INTO "public"."cities" ("cityid","cityname","countryid") VALUES
↪  (2,'Quezon City',13);
```

```
INSERT INTO "public"."cities" ("cityid","cityname","countryid") VALUES
↪ (3,'Philadelphia (PA)',22);


INSERT INTO "public"."cities" ("cityid","cityname","countryid") VALUES
↪ (4,'Jeddah',32);


INSERT INTO "public"."cities" ("cityid","cityname","countryid") VALUES
↪ (5,'Zhucheng',32);


INSERT INTO "public"."cities" ("cityid","cityname","countryid") VALUES
↪ (6,'Lagos',32);


INSERT INTO "public"."cities" ("cityid","cityname","countryid") VALUES
↪ (7,'Omdurman',34);


INSERT INTO "public"."cities" ("cityid","cityname","countryid") VALUES
↪ (8,'Donetsk',44);


INSERT INTO "public"."cities" ("cityid","cityname","countryid") VALUES
↪ (9,'Suqian',44);


INSERT INTO "public"."cities" ("cityid","cityname","countryid") VALUES
↪ (10,'Changsha',44);





INSERT INTO "public"."postalcodes"
↪ ("postalcodeid","postalcode","cityid") VALUES (1,'85745',8);


INSERT INTO "public"."postalcodes"
↪ ("postalcodeid","postalcode","cityid") VALUES (2,'21912',12);

```

```

INSERT INTO "public"."postalcodes" ("postalcodeid","postalcode","cityid") VALUES (3,'11650',12);


INSERT INTO "public"."postalcodes" ("postalcodeid","postalcode","cityid") VALUES (4,'14107',12);


INSERT INTO "public"."postalcodes" ("postalcodeid","postalcode","cityid") VALUES (5,'24983',20);


INSERT INTO "public"."postalcodes" ("postalcodeid","postalcode","cityid") VALUES (6,'16063',20);


INSERT INTO "public"."postalcodes" ("postalcodeid","postalcode","cityid") VALUES (7,'79001',27);


INSERT INTO "public"."postalcodes" ("postalcodeid","postalcode","cityid") VALUES (8,'37037',27);


INSERT INTO "public"."postalcodes" ("postalcodeid","postalcode","cityid") VALUES (9,'00085',30);


INSERT INTO "public"."postalcodes" ("postalcodeid","postalcode","cityid") VALUES (10,'99446',38);






INSERT INTO "public"."conference" ("confid","conftopic","startdate","enddate","postalcodeid","studentdiscount") VALUES (1,'Icon','2016-01-01','2016-01-02',4,0.25);


INSERT INTO "public"."conference" ("confid","conftopic","startdate","enddate","postalcodeid","studentdiscount") VALUES (2,'Java','2016-01-15','2016-01-16',4,0.7);
```

```sql


INSERT INTO "public"."conference"
↪ ("confid","conftopic","startdate","enddate","postalcodeid","studentdiscount")
↪ VALUES (3,'Javascript','2016-01-29','2016-01-30',10,0.3);


INSERT INTO "public"."conference"
↪ ("confid","conftopic","startdate","enddate","postalcodeid","studentdiscount")
↪ VALUES (4,'Elixir','2016-02-12','2016-02-13',10,0.5);


INSERT INTO "public"."conference"
↪ ("confid","conftopic","startdate","enddate","postalcodeid","studentdiscount")
↪ VALUES (5,'C','2016-02-26','2016-02-27',13,0);


INSERT INTO "public"."conference"
↪ ("confid","conftopic","startdate","enddate","postalcodeid","studentdiscount")
↪ VALUES (6,'C++','2016-03-11','2016-03-12',13,0.5);


INSERT INTO "public"."conference"
↪ ("confid","conftopic","startdate","enddate","postalcodeid","studentdiscount")
↪ VALUES (7,'C#','2016-03-25','2016-03-26',17,0.4);


INSERT INTO "public"."conference"
↪ ("confid","conftopic","startdate","enddate","postalcodeid","studentdiscount")
↪ VALUES (8,'.NET','2016-04-08','2016-04-09',17,0.1);


INSERT INTO "public"."conference"
↪ ("confid","conftopic","startdate","enddate","postalcodeid","studentdiscount")
↪ VALUES (9,'Elixir','2016-04-22','2016-04-23',20,0.18);


INSERT INTO "public"."conference"
↪ ("confid","conftopic","startdate","enddate","postalcodeid","studentdiscount")
↪ VALUES (10,'Scala','2016-05-06','2016-05-07',27,0.3);




```

```
INSERT INTO "public"."conferenceday"
↪ ("confid","confdate","daytopic","starttime","endtime","address","roomno","noseats
↪ VALUES (6,'2004-05-29','Pontiac','06:50:00','07:28:00','451 Serena
↪ Road','0.965',206,3.23);


INSERT INTO "public"."conferenceday"
↪ ("confid","confdate","daytopic","starttime","endtime","address","roomno","noseats
↪ VALUES (9,'2009-01-26','Infiniti','08:32:00','07:36:00','7912 N.
↪ Deerwood Avenue','5.629',192,76.32);


INSERT INTO "public"."conferenceday"
↪ ("confid","confdate","daytopic","starttime","endtime","address","roomno","noseats
↪ VALUES (15,'2011-06-28','Volkswagen','06:56:00','08:26:00','7 S.
↪ Riverside Plaza','3.065',202,59.34);


INSERT INTO "public"."conferenceday"
↪ ("confid","confdate","daytopic","starttime","endtime","address","roomno","noseats
↪ VALUES (20,'2007-03-23','Saturn','06:03:00','02:52:00','287
↪ Fairfield Road','9.362',196,0.70);


INSERT INTO "public"."conferenceday"
↪ ("confid","confdate","daytopic","starttime","endtime","address","roomno","noseats
↪ VALUES (26,'2017-11-04','Cadillac','05:22:00','06:12:00','811
↪ Grassmere Avenue','0.626',197,267.40);


INSERT INTO "public"."conferenceday"
↪ ("confid","confdate","daytopic","starttime","endtime","address","roomno","noseats
↪ VALUES (33,'2018-03-12','Isuzu','06:44:00','03:03:00','283 Sutter
↪ St','8.285',196,53.08);


INSERT INTO "public"."conferenceday"
↪ ("confid","confdate","daytopic","starttime","endtime","address","roomno","noseats
↪ VALUES (41,'2008-12-21','Mercedes-Benz','07:44:00','03:46:00','1
↪ Virginia Foothills Dr','2.951',201,130.55);


```

```
INSERT INTO "public"."conferenceday"
 ↪  ("confid","confdate","daytopic","starttime","endtime","address","roomno","noseats
 ↪  VALUES (42,'2012-07-02',NULL,'04:45:00','07:32:00','7004 Maxwell
 ↪  Ave.','6.720',191,287.07);


INSERT INTO "public"."conferenceday"
 ↪  ("confid","confdate","daytopic","starttime","endtime","address","roomno","noseats
 ↪  VALUES (50,'2015-10-26','Lincoln','07:13:00','04:32:00','57 Elmar
 ↪  Street','1.574',197,640.16);


INSERT INTO "public"."conferenceday"
 ↪  ("confid","confdate","daytopic","starttime","endtime","address","roomno","noseats
 ↪  VALUES (55,'2011-03-26','Honda','02:47:00','09:18:00','010 J.
 ↪  Carcione Way','4.492',209,26476.37);





INSERT INTO "public"."customers"
 ↪  ("customerid","customertype","address","postalcodeid","phone","login","password",
 ↪  VALUES (1,'company','7729 Copeland
 ↪  Street',10,'1-652-0096','Jeanne310','394171','thmpe@czwf.com');


INSERT INTO "public"."customers"
 ↪  ("customerid","customertype","address","postalcodeid","phone","login","password",
 ↪  VALUES (2,'person','1753 Palos Verdes
 ↪  Mall',10,'336-2120','Petra','768422','jlvis@ptlj.pl');


INSERT INTO "public"."customers"
 ↪  ("customerid","customertype","address","postalcodeid","phone","login","password",
 ↪  VALUES (3,'person','5 Edith Marie
 ↪  Drive',18,'002-6043','Cian2','573061','ncacp@kmlx.pl');


INSERT INTO "public"."customers"
 ↪  ("customerid","customertype","address","postalcodeid","phone","login","password",
 ↪  VALUES (4,'person','2966 North Fuller
 ↪  Avenue',18,'748-108-3066','Hugo0','051000','kooqi@uics.com');

```

```

INSERT INTO "public"."customers"
 ↪  ("customerid","customertype","address","postalcodeid","phone","login","password",
 ↪  VALUES (5,'person','281 Broomhouse
 ↪  Rd',18,'838-5099','Lotte','694962','vhlsp@prpa.com');


INSERT INTO "public"."customers"
 ↪  ("customerid","customertype","address","postalcodeid","phone","login","password",
 ↪  VALUES (6,'company','544 Ninth
 ↪  Avenue',20,'316-730-3862','Alvaro','056730','xgfhe@ttby.pl');


INSERT INTO "public"."customers"
 ↪  ("customerid","customertype","address","postalcodeid","phone","login","password",
 ↪  VALUES (7,'company','5 E. Plumb
 ↪  Lane',26,'1-014-9959','Jose4','405049','qcbvc@dvoo.com');


INSERT INTO "public"."customers"
 ↪  ("customerid","customertype","address","postalcodeid","phone","login","password",
 ↪  VALUES (8,'person','106
 ↪  Brookfields',30,'1-556-6056','George7','321194','msflq@hqpq.pl');


INSERT INTO "public"."customers"
 ↪  ("customerid","customertype","address","postalcodeid","phone","login","password",
 ↪  VALUES (9,'company','886 Springfield
 ↪  Ave',32,'357-573-6330','Coby12','256328','edmcp@cude.com');


INSERT INTO "public"."customers"
 ↪  ("customerid","customertype","address","postalcodeid","phone","login","password",
 ↪  VALUES (10,'company','8 Devonshire
 ↪  Rd',32,'368-728-1023','Jeanne6','577717','ubejl@yzxz.pl');





INSERT INTO "public"."individualcustomers"
 ↪  ("customerid","lastname","firstname") VALUES (10,'Malone','John');


```

```sql
INSERT INTO "public"."individualcustomers"
↪ ("customerid","lastname","firstname") VALUES (12,'Tudisco','Lucia');


INSERT INTO "public"."individualcustomers"
↪ ("customerid","lastname","firstname") VALUES (22,'Wood','Jace');


INSERT INTO "public"."individualcustomers"
↪ ("customerid","lastname","firstname") VALUES
↪ (28,'Overton','Thelma');


INSERT INTO "public"."individualcustomers"
↪ ("customerid","lastname","firstname") VALUES (31,'Brown','Zoe');


INSERT INTO "public"."individualcustomers"
↪ ("customerid","lastname","firstname") VALUES (37,'Dean','Marta');


INSERT INTO "public"."individualcustomers"
↪ ("customerid","lastname","firstname") VALUES
↪ (47,'Marra','Carolina');


INSERT INTO "public"."individualcustomers"
↪ ("customerid","lastname","firstname") VALUES (57,'Paddock','Lena');


INSERT INTO "public"."individualcustomers"
↪ ("customerid","lastname","firstname") VALUES (65,'Sanders','Gill');


INSERT INTO "public"."individualcustomers"
↪ ("customerid","lastname","firstname") VALUES (75,'Chwatal','Andy');





INSERT INTO "public"."pricetresholds" ("confid","until","discount")
↪ VALUES (6,'2001-01-12 04:48:00',0.5);

```

```sql

INSERT INTO "public"."pricetresholds" ("confid","until","discount")
↪  VALUES (7,'2013-05-11 07:08:00',0.1);


INSERT INTO "public"."pricetresholds" ("confid","until","discount")
↪  VALUES (9,'2001-04-10 02:54:00',0.44);


INSERT INTO "public"."pricetresholds" ("confid","until","discount")
↪  VALUES (17,'2009-05-27 00:23:00',0.2);


INSERT INTO "public"."pricetresholds" ("confid","until","discount")
↪  VALUES (23,'2004-08-26 03:58:00',0.1);


INSERT INTO "public"."pricetresholds" ("confid","until","discount")
↪  VALUES (26,'2010-05-06 07:11:00',0.22);


INSERT INTO "public"."pricetresholds" ("confid","until","discount")
↪  VALUES (36,'2012-01-28 05:33:00',0.3);


INSERT INTO "public"."pricetresholds" ("confid","until","discount")
↪  VALUES (38,'2005-05-24 09:08:00',0.1);


INSERT INTO "public"."pricetresholds" ("confid","until","discount")
↪  VALUES (47,'2008-03-04 03:58:00',0.4);


INSERT INTO "public"."pricetresholds" ("confid","until","discount")
↪  VALUES (55,'2004-02-01 07:23:00',0.29);





```

```
INSERT INTO "public"."workshop"
↪  ("workshopid","confid","confdate","workshoptopic","starttime","endtime","address'
↪  VALUES
↪  (1,31,'2005-01-12','nwv74Upp2SeR2IeCsq8i44AZS8Y8EaHyh2tQH','10:28:00','05:23:00',
↪  Reeves Avenue','8.921',198,75.10);


INSERT INTO "public"."workshop"
↪  ("workshopid","confid","confdate","workshoptopic","starttime","endtime","address'
↪  VALUES
↪  (2,31,'2005-01-12','xYbSTR4Rv21CUohXupkc','01:30:00','00:58:00','19
↪  Southwest Blvd','8.244',202,138.38);


INSERT INTO "public"."workshop"
↪  ("workshopid","confid","confdate","workshoptopic","starttime","endtime","address'
↪  VALUES
↪  (3,31,'2005-01-12','1YmQjXgPCBADtJD5umObThqUGzfGNgFxJpSojICwfKBfLSOXDXQeFs05TIuR'
↪  Main St.','1.226',210,01427.92);


INSERT INTO "public"."workshop"
↪  ("workshopid","confid","confdate","workshoptopic","starttime","endtime","address'
↪  VALUES
↪  (4,82,'2006-06-28','OZKj6ce2iCZUdeD8mi4I1W6yTARx0vgWEtewlGnaz3AeswUbKv1GAZU4ns7b6
↪  Francis Place','3.307',197,392.02);


INSERT INTO "public"."workshop"
↪  ("workshopid","confid","confdate","workshoptopic","starttime","endtime","address'
↪  VALUES
↪  (5,82,'2006-06-28','tMaX3UWNEMBYtOCT2hBEB1tof3phkVfMQFvsaW','08:42:00','06:21:00'
↪  E. Marions Rd','4.805',191,3691.56);


INSERT INTO "public"."workshop"
↪  ("workshopid","confid","confdate","workshoptopic","starttime","endtime","address'
↪  VALUES
↪  (6,99,'2011-11-16','iVCfc8wNFzT1dPjrL','04:43:00','06:27:00','24
↪  Vanderheck','5.824',191,638.11);


```

```sql
INSERT INTO "public"."workshop"
 ↪  ("workshopid","confid","confdate","workshoptopic","starttime","endtime","address'
 ↪  VALUES (7,99,'2011-11-16','U','07:23:00','10:34:00','1 Reeves
 ↪  Avenue','5.779',193,16.70);


INSERT INTO "public"."workshop"
 ↪  ("workshopid","confid","confdate","workshoptopic","starttime","endtime","address'
 ↪  VALUES
 ↪  (8,99,'2011-11-16','px7DXyrZmpKJ1Tfe5pzrGPm6Rj5H1','06:16:00','01:52:00','233
 ↪  Freeman St','9.130',193,905.28);


INSERT INTO "public"."workshop"
 ↪  ("workshopid","confid","confdate","workshoptopic","starttime","endtime","address'
 ↪  VALUES (9,72,'2011-11-16',NULL,'00:13:00','01:07:00','23 Greenheath
 ↪  Ave','3.568',203,62398.20);


INSERT INTO "public"."workshop"
 ↪  ("workshopid","confid","confdate","workshoptopic","starttime","endtime","address'
 ↪  VALUES
 ↪  (10,72,'2012-06-08','MN5bNqpDzVWbo1buHCkeOxo4sRtanu0DfGZmz00ZSRnEybdJe7Ew80KuuYdh
 ↪  S. Brockbank Drive','6.308',201,9715.35);





INSERT INTO "public"."companycustomers"
 ↪  ("customerid","companyname","fax") VALUES (2,'Saemto
 ↪  Bank','75987778');


INSERT INTO "public"."companycustomers"
 ↪  ("customerid","companyname","fax") VALUES (12,'BASF','19345190');


INSERT INTO "public"."companycustomers"
 ↪  ("customerid","companyname","fax") VALUES (17,'Alinca
 ↪  Systems','45616601');


```

```
INSERT INTO "public"."companycustomers"
↪ ("customerid","companyname","fax") VALUES (18,'Casco','44820801');


INSERT INTO "public"."companycustomers"
↪ ("customerid","companyname","fax") VALUES (23,'Riomme
↪ Bank','94403751');


INSERT INTO "public"."companycustomers"
↪ ("customerid","companyname","fax") VALUES (31,'Canberra
↪ Bank','66740568');


INSERT INTO "public"."companycustomers"
↪ ("customerid","companyname","fax") VALUES (37,'Brica','42265219');


INSERT INTO "public"."companycustomers"
↪ ("customerid","companyname","fax") VALUES (39,'Abonso
↪ Consultancy','75837760');


INSERT INTO "public"."companycustomers"
↪ ("customerid","companyname","fax") VALUES (46,'Cszerwinski
↪ Informatics','74214951');


INSERT INTO "public"."companycustomers"
↪ ("customerid","companyname","fax") VALUES (49,'Brica','77376376');






INSERT INTO "public"."conferencebookings"
↪ ("bookingid","confid","customerid","bookingtime") VALUES
↪ (1,11,7,'2000-07-31 09:27:00');


INSERT INTO "public"."conferencebookings"
↪ ("bookingid","confid","customerid","bookingtime") VALUES
↪ (2,11,7,'2011-04-12 04:45:00');

```

```
INSERT INTO "public"."conferencebookings"
↪ ("bookingid","confid","customerid","bookingtime") VALUES
↪ (3,11,7,'2003-08-14 03:50:00');


INSERT INTO "public"."conferencebookings"
↪ ("bookingid","confid","customerid","bookingtime") VALUES
↪ (4,17,10,'2016-11-02 06:40:00');


INSERT INTO "public"."conferencebookings"
↪ ("bookingid","confid","customerid","bookingtime") VALUES
↪ (5,24,10,'2002-10-09 01:52:00');


INSERT INTO "public"."conferencebookings"
↪ ("bookingid","confid","customerid","bookingtime") VALUES
↪ (6,24,10,'2009-10-14 02:50:00');


INSERT INTO "public"."conferencebookings"
↪ ("bookingid","confid","customerid","bookingtime") VALUES
↪ (7,24,19,'2004-04-27 08:49:00');


INSERT INTO "public"."conferencebookings"
↪ ("bookingid","confid","customerid","bookingtime") VALUES
↪ (8,25,19,'2001-06-02 03:25:00');


INSERT INTO "public"."conferencebookings"
↪ ("bookingid","confid","customerid","bookingtime") VALUES
↪ (9,25,19,'2015-05-17 02:15:00');


INSERT INTO "public"."conferencebookings"
↪ ("bookingid","confid","customerid","bookingtime") VALUES
↪ (10,34,29,'2001-05-21 09:26:00');
```

```sql
INSERT INTO "public"."conferenceparticipations" ("participationid","participantid","bookingid","studentid") VALUES (1,8,10,'2938003');


INSERT INTO "public"."conferenceparticipations" ("participationid","participantid","bookingid","studentid") VALUES (2,8,10,'2682818');


INSERT INTO "public"."conferenceparticipations" ("participationid","participantid","bookingid","studentid") VALUES (3,14,10,'5900140');


INSERT INTO "public"."conferenceparticipations" ("participationid","participantid","bookingid","studentid") VALUES (4,24,12,'7827506');


INSERT INTO "public"."conferenceparticipations" ("participationid","participantid","bookingid","studentid") VALUES (5,32,12,'1497311');


INSERT INTO "public"."conferenceparticipations" ("participationid","participantid","bookingid","studentid") VALUES (6,40,12,NULL);


INSERT INTO "public"."conferenceparticipations" ("participationid","participantid","bookingid","studentid") VALUES (7,44,20,NULL);


INSERT INTO "public"."conferenceparticipations" ("participationid","participantid","bookingid","studentid") VALUES (8,44,20,'4142737');


INSERT INTO "public"."conferenceparticipations" ("participationid","participantid","bookingid","studentid") VALUES (9,52,20,NULL);


INSERT INTO "public"."conferenceparticipations"
↪  ("participationid","participantid","bookingid","studentid") VALUES
↪  (10,52,28,NULL);





INSERT INTO "public"."daybookings"
↪  ("daybookingid","bookingid","confid","confdate","noseats","nostudents")
↪  VALUES (1,6,20,'2011-04-27',208,20);


INSERT INTO "public"."daybookings"
↪  ("daybookingid","bookingid","confid","confdate","noseats","nostudents")
↪  VALUES (2,6,20,'2013-02-07',203,24);


INSERT INTO "public"."daybookings"
↪  ("daybookingid","bookingid","confid","confdate","noseats","nostudents")
↪  VALUES (3,15,20,'2013-02-07',192,17);


INSERT INTO "public"."daybookings"
↪  ("daybookingid","bookingid","confid","confdate","noseats","nostudents")
↪  VALUES (4,23,53,'2018-06-13',202,29);


INSERT INTO "public"."daybookings"
↪  ("daybookingid","bookingid","confid","confdate","noseats","nostudents")
↪  VALUES (5,23,53,'2018-06-13',210,13);


INSERT INTO "public"."daybookings"
↪  ("daybookingid","bookingid","confid","confdate","noseats","nostudents")
↪  VALUES (6,26,53,'2018-06-13',190,30);


INSERT INTO "public"."daybookings"
↪  ("daybookingid","bookingid","confid","confdate","noseats","nostudents")
↪  VALUES (7,26,60,'2003-03-18',207,30);


```

```
INSERT INTO "public"."daybookings"
↪ ("daybookingid","bookingid","confid","confdate","noseats","nostudents")
↪ VALUES (8,34,93,'2003-03-18',202,17);


INSERT INTO "public"."daybookings"
↪ ("daybookingid","bookingid","confid","confdate","noseats","nostudents")
↪ VALUES (9,34,93,'2017-03-21',190,12);


INSERT INTO "public"."daybookings"
↪ ("daybookingid","bookingid","confid","confdate","noseats","nostudents")
↪ VALUES (10,44,15,'2000-03-30',193,29);





INSERT INTO "public"."dayparticipations"
↪ ("participationid","daybookingid") VALUES (11,436);


INSERT INTO "public"."dayparticipations"
↪ ("participationid","daybookingid") VALUES (21,4047);


INSERT INTO "public"."dayparticipations"
↪ ("participationid","daybookingid") VALUES (25,943);


INSERT INTO "public"."dayparticipations"
↪ ("participationid","daybookingid") VALUES (27,12226);


INSERT INTO "public"."dayparticipations"
↪ ("participationid","daybookingid") VALUES (34,10196);


INSERT INTO "public"."dayparticipations"
↪ ("participationid","daybookingid") VALUES (40,10393);


INSERT INTO "public"."dayparticipations"
↪ ("participationid","daybookingid") VALUES (48,10536);
```

```sql


INSERT INTO "public"."dayparticipations"
↪ ("participationid","daybookingid") VALUES (58,10554);


INSERT INTO "public"."dayparticipations"
↪ ("participationid","daybookingid") VALUES (62,4826);


INSERT INTO "public"."dayparticipations"
↪ ("participationid","daybookingid") VALUES (68,1732);





INSERT INTO "public"."payments"
↪ ("paymentid","bookingid","paymenttime","amount") VALUES
↪ (1,3,'2018-05-26 06:36:00',9152.85);


INSERT INTO "public"."payments"
↪ ("paymentid","bookingid","paymenttime","amount") VALUES
↪ (2,3,'2002-10-11 01:33:00',3.68);


INSERT INTO "public"."payments"
↪ ("paymentid","bookingid","paymenttime","amount") VALUES
↪ (3,3,'2003-04-04 00:06:00',75473.10);


INSERT INTO "public"."payments"
↪ ("paymentid","bookingid","paymenttime","amount") VALUES
↪ (4,4,'2012-05-17 10:26:00',619.84);


INSERT INTO "public"."payments"
↪ ("paymentid","bookingid","paymenttime","amount") VALUES
↪ (5,12,'2003-05-30 01:47:00',0575.09);


```

```
INSERT INTO "public"."payments"
↪ ("paymentid","bookingid","paymenttime","amount") VALUES
↪ (6,12,'2011-01-14 10:48:00',16.75);


INSERT INTO "public"."payments"
↪ ("paymentid","bookingid","paymenttime","amount") VALUES
↪ (7,12,'2018-12-19 09:48:00',43.15);


INSERT INTO "public"."payments"
↪ ("paymentid","bookingid","paymenttime","amount") VALUES
↪ (8,17,'2015-07-30 08:45:00',00.29);


INSERT INTO "public"."payments"
↪ ("paymentid","bookingid","paymenttime","amount") VALUES
↪ (9,17,'2013-11-09 09:53:00',0.24);


INSERT INTO "public"."payments"
↪ ("paymentid","bookingid","paymenttime","amount") VALUES
↪ (10,22,'2015-11-01 07:12:00',930.87);





INSERT INTO "public"."workshopbookings"
↪ ("daybookingid","workshopid","noseats","nostudents") VALUES
↪ (437,7,194,1);


INSERT INTO "public"."workshopbookings"
↪ ("daybookingid","workshopid","noseats","nostudents") VALUES
↪ (310,11,193,25);


INSERT INTO "public"."workshopbookings"
↪ ("daybookingid","workshopid","noseats","nostudents") VALUES
↪ (4159,12,203,17);


```

```
INSERT INTO "public"."workshopbookings"
↪  ("daybookingid","workshopid","noseats","nostudents") VALUES
↪  (12213,13,202,17);


INSERT INTO "public"."workshopbookings"
↪  ("daybookingid","workshopid","noseats","nostudents") VALUES
↪  (12225,312,197,6);


INSERT INTO "public"."workshopbookings"
↪  ("daybookingid","workshopid","noseats","nostudents") VALUES
↪  (12226,485,204,18);


INSERT INTO "public"."workshopbookings"
↪  ("daybookingid","workshopid","noseats","nostudents") VALUES
↪  (10195,621,193,6);


INSERT INTO "public"."workshopbookings"
↪  ("daybookingid","workshopid","noseats","nostudents") VALUES
↪  (10260,733,197,23);


INSERT INTO "public"."workshopbookings"
↪  ("daybookingid","workshopid","noseats","nostudents") VALUES
↪  (10392,738,193,6);


INSERT INTO "public"."workshopbookings"
↪  ("daybookingid","workshopid","noseats","nostudents") VALUES
↪  (4718,769,204,24);





INSERT INTO "public"."workshopparticipations"
↪  ("participationid","daybookingid","workshopid") VALUES
↪  (11,4046,4127);


```

```sql
INSERT INTO "public"."workshopparticipations"
↪  ("participationid","daybookingid","workshopid") VALUES
↪  (55,4437,4937);


INSERT INTO "public"."workshopparticipations"
↪  ("participationid","daybookingid","workshopid") VALUES
↪  (76,1747,9362);


INSERT INTO "public"."workshopparticipations"
↪  ("participationid","daybookingid","workshopid") VALUES
↪  (13,3174,10353);


INSERT INTO "public"."workshopparticipations"
↪  ("participationid","daybookingid","workshopid") VALUES
↪  (35,10162,10393);


INSERT INTO "public"."workshopparticipations"
↪  ("participationid","daybookingid","workshopid") VALUES
↪  (65,10976,2943);


INSERT INTO "public"."workshopparticipations"
↪  ("participationid","daybookingid","workshopid") VALUES
↪  (71,11117,3174);


INSERT INTO "public"."workshopparticipations"
↪  ("participationid","daybookingid","workshopid") VALUES
↪  (13,4127,10473);


INSERT INTO "public"."workshopparticipations"
↪  ("participationid","daybookingid","workshopid") VALUES
↪  (50,10393,5578);


INSERT INTO "public"."workshopparticipations"
↪  ("participationid","daybookingid","workshopid") VALUES
↪  (69,10872,10195);
```