

Projekt Bazy Danych - Konferencje

Wykonanie:

Łukasz Jezapkowicz

Bartłomiej Plewnia

Analiza wymagań

Wnioski z dokumentu projektowego oraz wywiadu przeprowadzonego z klientem

Konferencje

1. Konferencje mogą być jedno- lub kilkudniowe.
2. Klientami mogą być osoby indywidualne jak i firmy
3. Uczestnikami konferencji mogą być tylko osoby
4. Każdy uczestnik konferencji otrzymuje identyfikator imienny
5. Uczestnicy mogą zarejestrować się na dowolny dzień konferencji
6. Firma przy rejestracji rezerwuje odpowiednią ilość miejsc na każdy dzień
7. Istnieje ograniczona ilość miejsc na konferencje
8. Z konferencją związane są warsztaty

Warsztaty

1. Na warsztaty mogą się zapisać tylko uczestnicy konferencji w danym dniu
2. Kilka warsztatów może trwać jednocześnie
3. Każdy uczestnik może uczestniczyć tylko w jednych warsztatach w danym momencie
4. Istnieje ograniczona ilość miejsc na warsztaty
5. Niektóre warsztaty są płatne, a niektóre bezpłatne
6. Warsztaty mogą mieć różny czas trwania
7. Każdy warsztat może miećną ilość uczestników

Opłaty

1. Istnieje wiele progów cenowych dla konferencji w zależności od tego kiedy się dokona rezerwacji. Im szybciej dokona się rezerwacji tym większą użytkownik dostanie zniżkę
2. Każda konferencja może mieć różne progi cenowe
3. Istnieje zniżka dla studentów uczestniczących w konferencji. Każdy student musi przy rezerwacji wpisać nr swojej legitymacji studenckiej
4. Opłata za warsztaty jest stała (może jej nie być)
5. Po dokonaniu rezerwacji klient ma tydzień czasu na dokonanie opłaty. W przeciwnym wypadku rezerwacja jest anulowana
6. Płatności są tylko i wyłącznie atomowe

Raporty

1. Przed konferencją organizator otrzymuje listę identyfikatorów uczestników na każdy dzień konferencji oraz na każdy warsztat. Listy posiadają również informacje o płatnościach klientów
2. Organizator powinien mieć информацию o najbardziej aktywnych uczestnikach

Funkcje bazy danych

Funkcje dla organizatora

- a) Tworzenie nowej konferencji
 - Ustalenie daty konferencji
 - Ustalenie limitu miejsc na poszczególne dni konferencji
 - Ustalenie zniżki studenckiej oraz progów cenowych
- b) Tworzenie nowego warsztatu do konferencji
 - Ustalenie daty warsztatu
 - Ustalenie limitu miejsc
 - Ustalenie ceny warsztatu
- c) Wyświetlenie listy uczestników danej konferencji
 - Organizator może wyświetlić listę uczestników danej konferencji, którzy opłacili swoją rezerwację
- d) Wyświetlenie listy uczestników danego warsztatu
 - Organizator może wyświetlić listę uczestników danego warsztatu, którzy opłacili swoją rezerwację
- e) Wyświetlanie statystyk najbardziej aktywnych uczestników
 - Organizator może wyświetlić listę uczestników, którzy wzięli udział w największej liczbie konferencji i/lub warsztatów
- f) Wyświetlanie konferencji z największym procentem zapełnienia
 - Organizator może wyświetlić listę konferencji, na których współczynnik zapełniania był największy
- g) Wyświetlanie warsztatów z największym procentem zapełnienia
 - Organizator może wyświetlić listę warsztatów, w których współczynnik zapełniania był największy

Funkcje dla klienta indywidualnego

- h) Rejestracja
 - Podanie podstawowych danych (imienia, ulicy, miasta, nr telefonu, e-maila) uczestnika
- i) Wyświetlenie dostępnych konferencji
 - Wyświetlenie dostępnych miejsc
 - Wyświetlenie ceny tych konferencji

- j) Wyświetlenie dostępnych warsztatów
 - Wyświetlenie dostępnych miejsc
 - Wyświetlenie ceny tych warsztatów
- k) Rezerwacja konferencji
 - Wybranie dostępnej konferencji, następnie dni konferencji
 - Możliwość rezerwacji warsztatów w dni konferencji, w których dana osoba uczestniczy
- l) Uzupełnienie danych z przypisanymi dniami konferencji oraz warsztatami
 - Możliwość podania numeru legitymacji studenckiej w celu skorzystania ze zniżki studenckiej
- m) Wgląd w koszty zarezerwowanych wydarzeń
 - Wyświetlenie sumarycznej ceny jaką klient musi zapłacić
- n) Opłata rezerwacji

Funkcje dla klienta zbiorowego - firmy

- o) Rejestracja
 - Podanie podstawowych danych (imienia, ulicy, miasta, nr telefonu, e-maila) firmy
- p) Wyświetlenie dostępnych konferencji
 - Wyświetlenie dostępnych miejsc
 - Wyświetlenie ceny tych konferencji
- q) Wyświetlenie dostępnych warsztatów
 - Wyświetlenie dostępnych miejsc
 - Wyświetlenie ceny tych warsztatów
- r) Rezerwacja konferencji dla danej ilości osób
 - Wybranie dostępnej konferencji, następnie dni konferencji
 - Możliwość rezerwacji warsztatów w dni konferencji, w których dana osoba uczestniczy
- s) Uzupełnienie danych osób z przypisanymi dniami konferencji oraz warsztatami
 - Możliwość zaznaczenia uczestników, którzy mogą skorzystać ze zniżki studenckiej podając ich numery legitymacji studenckiej
- t) Wgląd w koszty zarezerwowanych wydarzeń
 - Wyświetlenie sumarycznej ceny jaką firma musi zapłacić
- r) Opłata rezerwacji

Funkcje uczestnika - osoba prywatna lub jeden z uczestników podany przez firmę

- s) Wyświetlenie unikalnego identyfikatora imiennego
 - Wyświetlenie unikalnego identyfikatora imiennego (oraz firmy, gdy uczestnik jest z firmy)
- t) Wyświetlenie zarezerwowanych konferencji/warsztatów
 - Wyświetlenie zarezerwowanych (opłaconych) konferencji oraz warsztatów, w których uczestnik weźmie udział

Funkcje systemu

- u) Usuwanie rezerwacji, które nie zostały opłacone w terminie
 - Każdego dnia system usuwa wszystkie zamówienia, które nie zostały opłacone w terminie tygodnia od złożenia rezerwacji
- v) Usuwanie grupowych rezerwacji, które nie zostały wypełnione w terminie
 - Każdego dnia system usuwa wszystkie grupowe zamówienia, które nie zostały uzupełnione odpowiednimi danymi w terminie dwóch tygodni do konferencji
- w) Wyliczanie sumarycznej ceny jaką klient indywidualny/zbiorowy musi zapłacić
- y) Obliczenie ilości miejsc dostępnych na dni konferencji oraz warsztaty

Diagram bazy danych ER

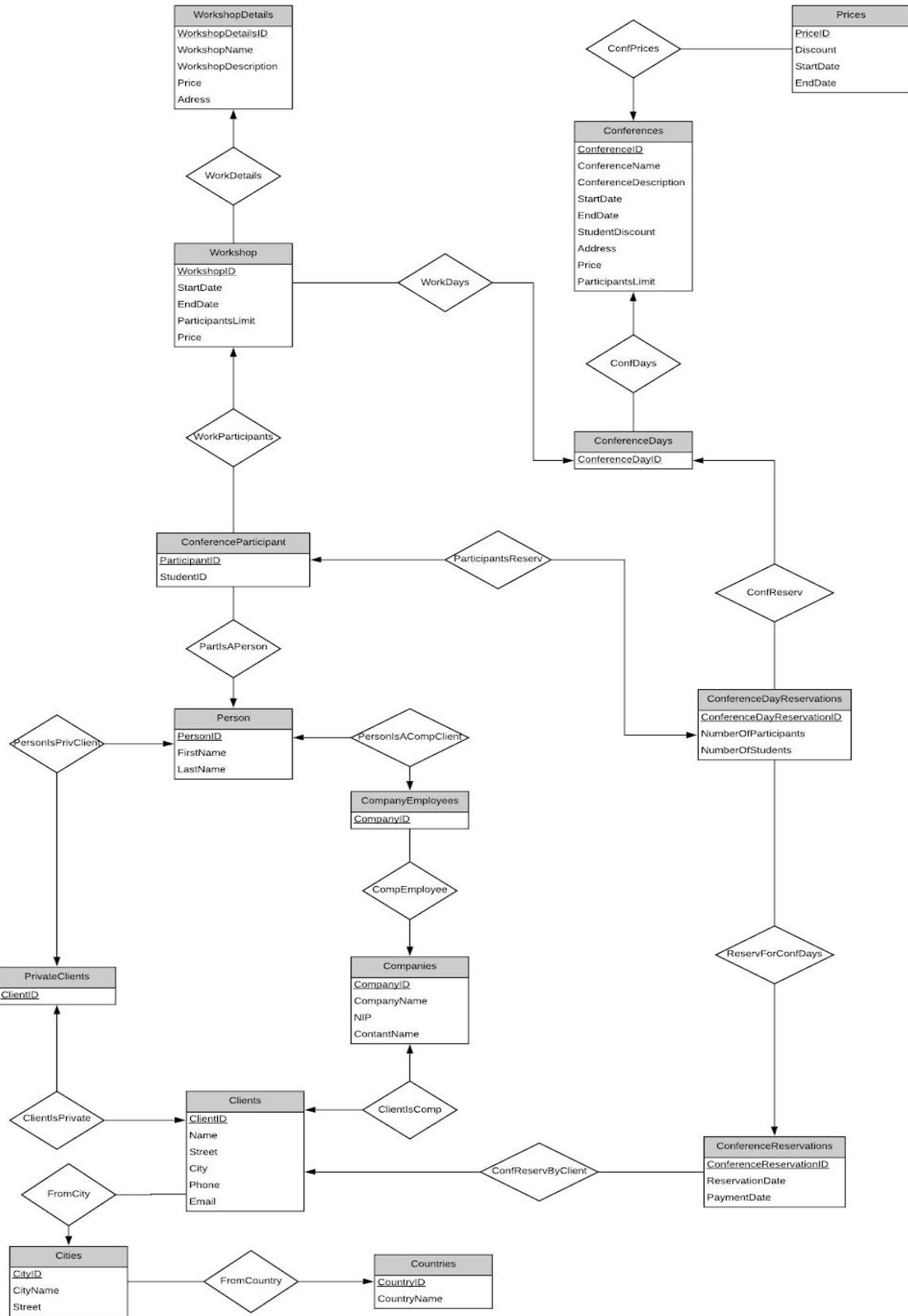
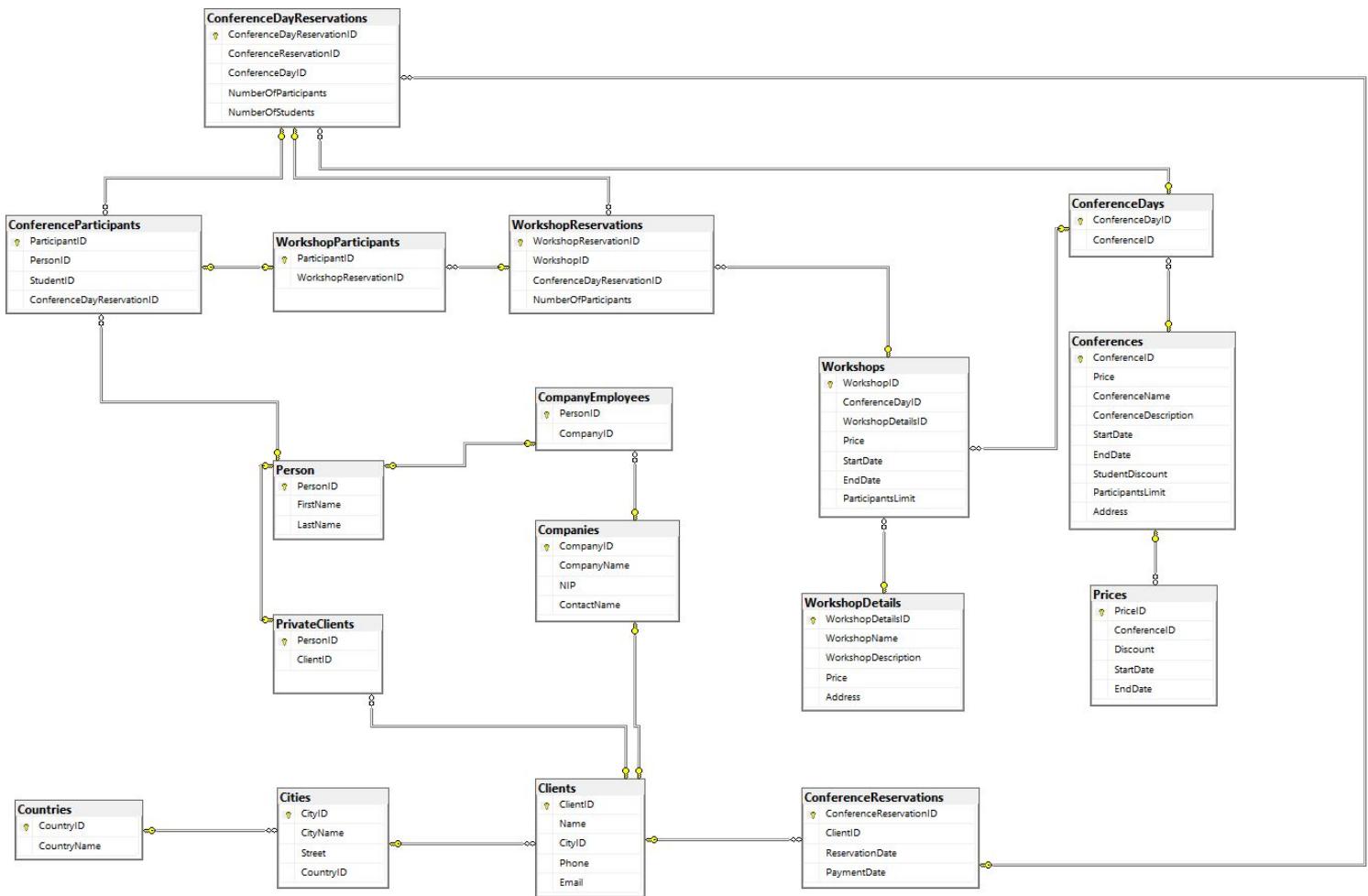


Diagram bazy danych



AD. 1

AD. 2

Tabele w bazie danych

Countries - przechowuje informacje o krajach klientów

- CountryID - Identyfikator kraju, wartość autoinkrementowana przez system (klucz główny)
- CountryName - nazwa kraju

Polecenie tworzące tabelę:

```
Create table Countries (
    CountryID int IDENTITY(1,1) PRIMARY KEY,
    CountryName varchar(30),
    UNIQUE (CountryName)
)
```

Cities - przechowuje informacje o miastach klientów

- CityID - Identyfikator kraju, wartość autoinkrementowana przez system (klucz główny)
- CityName - nazwa miasta
- Street - ulica
- CountryID - Identyfikator kraju (klucz obcy)

Polecenie tworzące tabelę:

```
Create table Cities (
    CityID int IDENTITY(1,1) PRIMARY KEY,
    CityName varchar(30),
    CountryID INT,
    UNIQUE(CityName),
    FOREIGN KEY(CountryID) References Countries(CountryID)
)
```

Clients - przechowuje informacje o klientach korzystających z usług systemu

- ClientID - Identyfikator klienta, wartość autoinkrementowana przez system (klucz główny)
- Name - Nazwa klienta
- CityID - Identyfikator miasta (klucz obcy)
- Phone - Numer telefonu klienta, musi zawierać +
- Email - Adres email klienta, musi zawierać @

Polecenie tworzące tabelę:

```
Create table Clients (
    ClientID int IDENTITY(1,1) PRIMARY KEY,
    Name varchar(30) NOT NULL,
    CityID int NOT NULL,
    Phone char(12) NOT NULL,
    Email varchar(30) NOT NULL,
    UNIQUE>Email,
    UNIQUE>Phone,
    CONSTRAINT CC_Phone CHECK (Phone like '%+%'),
    CONSTRAINT CC_Phone2 CHECK (Phone not like '%[^0-9+]%'),
    CONSTRAINT CC_Email CHECK (Email like '%@%'),
    FOREIGN KEY(CityID) References Cities(CityID)
);
```

Companies - przechowuje informacje o firmach

- CompanyID - Identyfikator firmy (klucz główny oraz klucz obcy)
- CompanyName - Nazwa firmy
- NIP - NIP firmy
- ContactName - Kontaktowa nazwa firmy

Polecenie tworzące tabelę:

```
Create table Companies (
    CompanyID int PRIMARY KEY,
    CompanyName varchar(30) NOT NULL,
    NIP char(10) NOT NULL,
    ContactName varchar(30) NOT NULL,
    UNIQUE(CompanyName),
    UNIQUE(NIP),
    FOREIGN KEY(CompanyID) REFERENCES Clients(ClientID)
);
```

Person - przechowuje informacje o osobach biorących udział w wydarzeniach

- PersonID - Identyfikator osoby, wartość autoinkrementowana przez system (klucz główny)
- FirstName - Imię osoby
- LastName - Nazwisko osoby

Polecenie tworzące tabelę:

```
Create table Person (
    PersonID int IDENTITY(1,1) PRIMARY KEY,
    FirstName varchar(30),
    LastName varchar(30)
);
```

PrivateClients - przechowuje informacje o klientach indywidualnych

- PersonID - Identyfikator osoby (klucz główny oraz klucz obcy)
- ClientID - Identyfikator klienta (klucz obcy)

Polecenie tworzące tabelę:

```
Create table PrivateClients (
    PersonID int PRIMARY KEY,
    ClientID int NOT NULL,
    FOREIGN KEY(ClientID) REFERENCES Clients(ClientID),
    FOREIGN KEY(PersonID) REFERENCES Person(PersonID)
);
```

CompanyEmployees - przechowuje informacje o pracownikach firmy biorących udział w wydarzeniach

- PersonID - Identyfikator osoby (klucz główny oraz klucz obcy)
- CompanyID - Identyfikator firmy (klucz obcy)

Polecenie tworzące tabelę:

```
Create table CompanyEmployees(
    PersonID int PRIMARY KEY,
    CompanyID int NOT NULL,
    FOREIGN KEY(CompanyID) REFERENCES Companies(CompanyID),
    FOREIGN KEY(PersonID) REFERENCES Person(PersonID)
);
```

ConferenceReservations - przechowuje informacje o rezerwacjach składanych przez klientów systemu

- ConferenceReservationID - Identyfikator rezerwacji konferencji, wartość autoinkrementowana przez system (klucz główny)
- ClientID - Identyfikator klienta (klucz obcy)
- ReservationDate - data złożenia rezerwacji konferencji
- PaymentDate - data zapłaty za rezerwację, musi być późniejsza niż ReservationDate

Polecenie tworzące tabelę:

```
Create table ConferenceReservations(  
    ConferenceReservationID int IDENTITY(1,1) PRIMARY KEY,  
    ClientID int NOT NULL,  
    ReservationDate datetime DEFAULT GETDATE(),  
    PaymentDate datetime,  
    FOREIGN KEY(ClientID) REFERENCES Clients(ClientID),  
    CONSTRAINT CCR_PaymentDate CHECK (ISNULL(PaymentDate,ReservationDate)>=ReservationDate)  
);
```

Conferences - przechowuje informacje o poszczególnych konferencjach

- ConferenceID - Identyfikator konferencji, wartość autoinkrementowana przez system (klucz główny)
- Price - Cena uczestnictwa w jednym dniu konferencji, musi być większa od 0
- ConferenceName - Nazwa konferencji
- ConferenceDescription - Opis konferencji
- StartDate - data rozpoczęcia konferencji (data pierwszego dnia)
- EndDate - data zakończenia konferencji (data ostatniego dnia), musi być późniejsza niż StartDate
- StudentDiscount - zniżka studencka (może być 0), musi zawierać się między 0 a 1
- ParticipantsLimit - limit uczestników w jednym dniu konferencji, musi być większy od 0
- Address - adres miejsca, w którym odbywa się konferencja

Polecenie tworzące tabelę:

```
Create table Conferences(  
    ConferenceID int IDENTITY(1,1) PRIMARY KEY,  
    Price money NOT NULL,  
    ConferenceName varchar(30) NOT NULL,  
    ConferenceDescription varchar(150) NOT NULL,  
    StartDate datetime NOT NULL,  
    EndDate datetime NOT NULL,  
    StudentDiscount decimal(3,3) DEFAULT 0,  
    ParticipantsLimit int NOT NULL,  
    Address varchar(60) NOT NULL,  
    CONSTRAINT CCO_Price CHECK (Price > 0),  
    CONSTRAINT CCO_EndDate CHECK (EndDate>StartDate),  
    CONSTRAINT CCO_StudentDiscount CHECK (StudentDiscount between 0 and 1),  
    CONSTRAINT CCO_ParticipantsLimit CHECK (ParticipantsLimit > 0)  
);
```

Prices - przechowuje informacje o zniżkach związanych z datą rezerwacji konferencji
→PriceID - Identyfikator ceny, wartość autoinkrementowana przez system (klucz główny)
→ConferenceID - Identyfikator konferencji (klucz obcy)
→Discount - wartość zniżki, musi zawierać się między 0 a 1
→StartDate - data, kiedy zaczyna obowiązywać dana zniżka
→EndDate - data, kiedy kończy obowiązywać dana zniżka, musi być późniejsza niż
StartDate

Polecenie tworzące tabelę:

```
Create table Prices(
    PriceID int IDENTITY(1,1) PRIMARY KEY,
    ConferenceID int NOT NULL,
    Discount decimal(3,3) DEFAULT 0,
    StartDate datetime NOT NULL,
    EndDate datetime NOT NULL,
    FOREIGN KEY (ConferenceID) REFERENCES Conferences(ConferenceID),
    CONSTRAINT CP_Discount CHECK (Discount between 0 and 1),
    CONSTRAINT CP_EndDate CHECK (EndDate>StartDate)
);
```

ConferenceDays - przechowuje informacje o poszczególnych dniach konferencji

→ConferenceDayID - Identyfikator dnia konferencji, wartość autoinkrementowana przez
system (klucz główny)
→ConferenceID - Identyfikator konferencji (klucz obcy)

Polecenie tworzące tabelę:

```
Create table ConferenceDays(
    ConferenceDayID int IDENTITY(1,1) PRIMARY KEY,
    ConferenceID int NOT NULL,
    FOREIGN KEY (ConferenceID) REFERENCES Conferences(ConferenceID)
);
```

WorkshopDetails - przechowuje informacje słownikowe o poszczególnych warsztatach

→WorkshopDetailsID - Identyfikator szczegółów dotyczących warsztatu, wartość
autoinkrementowana przez system (klucz główny)
→WorkshopName - Nazwa warsztatu
→WorkshopDescription - Opis warsztatu
→Price - Cena warsztatu, musi być większa od 0
→Address - Adres miejsca, w którym odbywa się warsztat

Polecenie tworzące tabelę:

```
Create table WorkshopDetails(
    WorkshopDetailsID int IDENTITY(1,1) PRIMARY KEY,
    WorkshopName varchar(30) NOT NULL,
    WorkshopDescription varchar(150) NOT NULL,
    Price money NOT NULL,
    Address varchar(60) NOT NULL,
    CONSTRAINT CWD_Price CHECK (Price > 0)
);
```

Workshops - przechowuje informacje o poszczególnych warsztatach

- WorkshopID - Identyfikator warsztatu, wartość autoinkrementowana przez system (klucz główny)
- ConferenceDayID - Identyfikator konferencji (klucz obcy)
- WorkshopDetailsID - Identyfikator szczegółów dotyczących warsztatu (klucz obcy)
- Price - Cena warsztatu, musi być większa od 0
- StartDate - Data rozpoczęcia warsztatu
- EndDate - Data zakończenia warsztatu, musi być późniejsza niż StartDate
- ParticipantsLimit - limit uczestników warsztatu, musi być większy od 0

Polecenie tworzące tabelę:

```
Create table Workshops(  
    WorkshopID int IDENTITY(1,1) PRIMARY KEY,  
    ConferenceDayID int NOT NULL,  
    WorkshopDetailsID int NOT NULL,  
    Price money NOT NULL,  
    StartDate datetime NOT NULL,  
    EndDate datetime NOT NULL,  
    ParticipantsLimit int NOT NULL,  
    FOREIGN KEY (ConferenceDayID) REFERENCES ConferenceDays(ConferenceDayID),  
    FOREIGN KEY (WorkshopDetailsID) REFERENCES WorkshopDetails(WorkshopDetailsID),  
    CONSTRAINT CW_Price CHECK (Price > 0),  
    CONSTRAINT CW_EndDate CHECK (EndDate>StartDate),  
    CONSTRAINT CW_ParticipantsLimit CHECK (ParticipantsLimit > 0)  
)
```

ConferenceDayReservations - przechowuje informacje o rezerwacjach poszczególnych dni konferencji

- ConferenceDayReservationID - Identyfikator rezerwacji dnia konferencji, wartość autoinkrementowana przez system (klucz główny)
- ConferenceReservationID - Identyfikator rezerwacji konferencji (klucz obcy)
- ConferenceDayID - Identyfikator dnia konferencji (klucz obcy)
- NumberOfParticipants - liczba uczestników bez zniżki, musi być nieujemna
- NumberOfStudents - liczba uczestników, którzy są studentami, musi być nieujemna

Polecenie tworzące tabelę:

```
Create table ConferenceDayReservations(  
    ConferenceDayReservationID int IDENTITY(1,1) PRIMARY KEY,  
    ConferenceReservationID int NOT NULL,  
    ConferenceDayID int NOT NULL,  
    NumberOfParticipants int NOT NULL,  
    NumberOfStudents int NOT NULL,  
    FOREIGN KEY (ConferenceReservationID) REFERENCES ConferenceReservations(ConferenceReservationID),  
    FOREIGN KEY (ConferenceDayID) REFERENCES ConferenceDays(ConferenceDayID),  
    CONSTRAINT CDDR_NumberOfParticipants CHECK (NumberOfParticipants >= 0),  
    CONSTRAINT CDDR_NumberOfStudents CHECK (NumberOfStudents >= 0)  
)
```

ConferenceParticipants - przechowuje informacje o uczestnikach konferencji

→ ParticipantID - Identyfikator uczestnika, wartość autoinkrementowana przez system (klucz główny)

→ PersonID - Identyfikator osoby (klucz obcy)

→ ConferenceDayReservationID - Identyfikator rezerwacji dnia konferencji (klucz obcy)

→ StudentID - Numer legitymacji studenckiej

Polecenie tworzące tabelę:

```
Create table ConferenceParticipants(
    ParticipantID int IDENTITY(1,1) PRIMARY KEY,
    PersonID int NOT NULL,
    StudentID int,
    ConferenceDayReservationID int NOT NULL,
    FOREIGN KEY (PersonID) REFERENCES Person(PersonID),
    FOREIGN KEY (ConferenceDayReservationID) REFERENCES ConferenceDayReservations(ConferenceDayReservationID)
);
```

WorkshopReservations - przechowuje informacje o rezerwacjach poszczególnych warsztatów

→ WorkshopReservationID - Identyfikator rezerwacji warsztatu, wartość autoinkrementowana przez system (klucz główny)

→ WorkshopID - Identyfikator warsztatu (klucz obcy)

→ ConferenceDayReservationID - Identyfikator rezerwacji dnia konferencji (klucz obcy)

→ NumberOfParticipants - liczba uczestników, musi być większa od 0

Polecenie tworzące tabelę:

```
Create table WorkshopReservations(
    WorkshopReservationID int IDENTITY(1,1) PRIMARY KEY,
    WorkshopID int NOT NULL,
    ConferenceDayReservationID int NOT NULL,
    NumberOfParticipants int NOT NULL,
    FOREIGN KEY (WorkshopID) REFERENCES Workshops(WorkshopID),
    FOREIGN KEY (ConferenceDayReservationID) REFERENCES ConferenceDayReservations(ConferenceDayReservationID),
    CONSTRAINT CWR_NumberOfParticipants CHECK (NumberOfParticipants > 0)
);
```

WorkshopParticipants - przechowuje informacje o uczestnikach warsztatów

→ ParticipantID - Identyfikator uczestnika (klucz główny)

→ DayParticipantID - Identyfikator uczestnika konferencji (klucz obcy)

→ WorkshopReservationID - Identyfikator rezerwacji warsztatu (klucz obcy)

Polecenie tworzące tabelę:

```
CREATE table WorkshopParticipants(
    ParticipantID int IDENTITY(1,1) PRIMARY KEY,
    DayParticipantID INT NOT NULL,
    WorkshopReservationID int NOT NULL,
    FOREIGN KEY (WorkshopReservationID) REFERENCES WorkshopReservations(WorkshopReservationID),
    FOREIGN KEY (DayParticipantID) REFERENCES ConferenceParticipants(ParticipantID)
);
```

Widoki

ConferenceDaysInfo - zwraca spis dni konferencji wraz z ilością zajętych/wolnych miejsc

```
Create view ConferenceDaysInfo as
Select CD.ConferenceDayID,C.ConferenceID,C.ConferenceName,C.ParticipantsLimit,
(SUM(ISNULL(CDR.NumberOfParticipants,0))+SUM(ISNULL(CDR.NumberOfStudents,0))) as PlacesTaken,
(C.ParticipantsLimit - (SUM(ISNULL(CDR.NumberOfParticipants,0))+
SUM(ISNULL(CDR.NumberOfStudents,0)))) as PlacesFree
From Conferences as C JOIN ConferenceDays as CD on C.ConferenceID=CD.ConferenceID
JOIN ConferenceDayReservations as CDR on CD.ConferenceDayID=CDR.ConferenceDayID
Group by CD.ConferenceDayID,C.ConferenceID,C.ConferenceName,C.ParticipantsLimit
Go
```

ConferenceDayParticipants - zwraca spis uczestników dni konferencji wraz z ich danymi

```
Create view ConferenceDayParticipants as
Select P.PersonID,P.FirstName, P.LastName,CD.ConferenceDayID, CD.ConferenceID, CP.ParticipantID
From ConferenceDays as CD JOIN ConferenceDayReservations as CDR
on CD.ConferenceDayID=CDR.ConferenceDayID
JOIN ConferenceParticipants as CP on CP.ConferenceDayReservationID=CDR.ConferenceDayReservationID
JOIN Person as P on P.PersonID=CP.PersonID
Go
```

ListedConferenceParticipants - zwraca spis uczestników konferencji wraz z ich danymi oraz informacją w ilu dniach poszczególnych konferencji uczestniczyli

```
CREATE view ListedConferenceParticipants as
Select CP.ParticipantID,P.PersonID,P.FirstName,P.LastName,C.ConferenceID, C.ConferenceName,
(DATEDIFF(DAY,C.StartDate,C.EndDate)+1) as NumberOfDays,
(Select count(*) from ConferenceDays as CD2 JOIN ConferenceDayReservations as CDR2
on CD2.ConferenceDayID=CDR2.ConferenceDayID JOIN ConferenceParticipants as CP2
on CP2.ConferenceDayReservationID=CDR2.ConferenceDayReservationID
Where CD2.ConferenceID=C.ConferenceID and CP2.ParticipantID=CP.ParticipantID) as DaysParticipated
From Conferences as C JOIN ConferenceDays as CD on C.ConferenceID=CD.ConferenceID
JOIN ConferenceDayReservations as CDR
on CD.ConferenceDayID=CDR.ConferenceDayID
JOIN ConferenceParticipants as CP on CP.ConferenceDayReservationID=CDR.ConferenceDayReservationID
JOIN Person as P on P.PersonID=CP.PersonID
GO
```

AvalaibleConferenceDays - zwraca spis dni konferencji, na które istnieją jeszcze wolne miejsca, wraz z informacją ile wolnych miejsc pozostało

```
Create view AvalaibleConferenceDays as
Select CD.ConferenceDayID, CD.ConferenceID, C.ParticipantsLimit,
(SUM(ISNULL(CDR.NumberOfParticipants,0))+SUM(ISNULL(CDR.NumberOfStudents,0))) as PlacesTaken,
(C.ParticipantsLimit - (SUM(ISNULL(CDR.NumberOfParticipants,0))+
SUM(ISNULL(CDR.NumberOfStudents,0)))) as PlacesFree
From ConferenceDays as CD JOIN Conferences as C on C.ConferenceID=CD.ConferenceID
JOIN ConferenceDayReservations as CDR on CDR.ConferenceDayID=CD.ConferenceDayID
Where C.ParticipantsLimit-(Select sum(CDR2.NumberOfParticipants+CDR2.NumberOfStudents)
from ConferenceDayReservations as CDR2
Where CDR.ConferenceDayID=CDR2.ConferenceDayID) > 0
Group by CD.ConferenceDayID, CD.ConferenceID, C.ParticipantsLimit
Go
```

WorkshopInfo - zwraca spis warsztatów wraz z informacją na temat zajętych/wolnych miejsc

```
Create view WorkshopInfo as
Select W.WorkshopID,W.ConferenceDayID,WD.WorkshopName, W.Price, W.ParticipantsLimit,
SUM(ISNULL(WR.NumberOfParticipants,0)) as PlacesTaken,
W.ParticipantsLimit - SUM(ISNULL(WR.NumberOfParticipants,0)) as PlacesFree
From Workshops as W JOIN WorkshopDetails as WD on W.WorkshopDetailsID=WD.WorkshopDetailsID
JOIN WorkshopReservations as WR on WR.WorkshopID=W.WorkshopID
Group by W.WorkshopID,W.ConferenceDayID,WD.WorkshopName, W.Price, W.ParticipantsLimit
Go
```

ListedWorkshopParticipants - zwraca spis uczestników warsztatów wraz z informacją ile spędzili czasu na poszczególnych warsztatach

```
CREATE view ListedWorkshopParticipants as
Select CP.ParticipantID,P.PersonID,P.FirstName,W.WorkshopID, WD.WorkshopName,
(DATEDIFF(hour,W.StartDate,W.EndDate)) as NumberOfHours
From Workshops as W JOIN WorkshopDetails as WD on W.WorkshopDetailsID=WD.WorkshopDetailsID
JOIN WorkshopReservations as WR on W.WorkshopID=WR.WorkshopID
JOIN WorkshopParticipants as WP on WP.WorkshopReservationID=WR.WorkshopReservationID
JOIN ConferenceParticipants as CP on WP.ParticipantID=CP.ParticipantID
JOIN Person as P on CP.PersonID=P.PersonID
Go
```

AvalaibleWorkshops - zwraca spis warsztatów, na które istnieją jeszcze wolne miejsca, wraz z ilością wolnych miejsc

```
Create view AvalaibleWorkshops as
Select W.WorkshopID,W.ConferenceDayID,WD.WorkshopName, W.Price, W.ParticipantsLimit,
SUM(ISNULL(WR.NumberOfParticipants,0)) as PlacesTaken,
W.ParticipantsLimit - SUM(ISNULL(WR.NumberOfParticipants,0)) as PlacesFree
From Workshops as W JOIN WorkshopDetails as WD on W.WorkshopDetailsID=WD.WorkshopDetailsID
JOIN WorkshopReservations as WR on WR.WorkshopID=W.WorkshopID
Where W.ParticipantsLimit - (Select sum(WR2.NumberOfParticipants) From
WorkshopReservations as WR2 where WR.WorkshopID=WR2.WorkshopID) > 0
Group by W.WorkshopID,W.ConferenceDayID,WD.WorkshopName, W.Price, W.ParticipantsLimit
Go
```

MostPopularConferences - zwraca 10 konferencji, w których wystąpił największy procent zapełnienia uczestników

```
Create view MostPopularConferenceDays as
Select CD.ConferenceDayID, CD.ConferenceID, C.ConferenceName, C.ParticipantsLimit,
(Select sum(ISNULL(CDR.NumberOfParticipants,0) + ISNULL(CDR.NumberOfStudents,0)))
as PlacesTaken,
((Select sum(ISNULL(CDR.NumberOfParticipants,0))+SUM(ISNULL(CDR.NumberOfStudents,0)))/
CAST(C.ParticipantsLimit as Decimal (9,2))) as Fill_Factor
From ConferenceDays as CD JOIN ConferenceDayReservations as CDR on CDR.ConferenceDayID=CD.ConferenceDayID
JOIN Conferences AS C ON C.ConferenceID=CD.ConferenceID
Group by CD.ConferenceDayID, CD.ConferenceID, C.ConferenceName, C.ParticipantsLimit
GO
```

MostPopularWorkshops - zwraca 10 warsztatów, w których wystąpił największy procent zapełnienia uczestników

```
CREATE view MostPopularWorkshops as
Select W.WorkshopID,WD.WorkshopName,W.ParticipantsLimit,
(Select sum(ISNULL(WR.NumberOfParticipants,0))) as PlacesTaken,
(Select sum(ISNULL(WR.NumberOfParticipants,0))/CAST(W.ParticipantsLimit as Decimal(9,2))as Fill_Factor
From Workshops as W JOIN WorkshopDetails as WD on W.WorkshopDetailsID=WD.WorkshopDetailsID
JOIN WorkshopReservations as WR on WR.WorkshopID=W.WorkshopID
Group by W.WorkshopID,WD.WorkshopName,W.ParticipantsLimit
Go
```

MostFrequentParticipants - zwraca 10 uczestników, którzy uczestniczyli w największej liczbie dni konferencji oraz warsztatach

```
CREATE view MostFrequentParticipants as
SELECT P.FirstName, P.LastName, ((Select count(*) from ConferenceParticipants as CP2 JOIN dbo.Person AS P2 ON P2.PersonID = CP2.PersonID
WHERE P2.FirstName=P.FirstName AND P2.LastName=P.LastName) +
(Select count(*) from WorkshopParticipants as WP JOIN dbo.ConferenceParticipants AS CP2 ON CP2.ParticipantID = WP.DayParticipantID
JOIN dbo.Person AS P2 ON P2.PersonID = CP2.PersonID WHERE P2.FirstName=P.FirstName AND P2.LastName=P.LastName)) as TotalParticipations
From ConferenceParticipants as CP JOIN Person as P on P.PersonID=CP.PersonID
Group by P.FirstName, P.LastName
Go
```

UnfilledBookings - zwraca rezerwacje, które nie zostały jeszcze wypełnione, ale nie minęły jeszcze 2 tygodnie od złożenia rezerwacji

```
Create view UnfilledBookings as
Select P.PersonID, P.FirstName, P.LastName, CDR.ConferenceDayID,
CR.ConferenceReservationID, CR.ClientID
From Person as P JOIN ConferenceParticipants as CP on P.PersonID=CP.PersonID
JOIN ConferenceDayReservations as CDR
on CDR.ConferenceDayReservationID=CP.ConferenceDayReservationID
JOIN ConferenceReservations as CR on CR.ConferenceReservationID=CDR.ConferenceReservationID
Where P.FirstName is NULL and P.LastName is NULL and
DATEDIFF(day,CR.ReservationDate,getdate())<=14
Go
```

BookingsPaidAfterDeadline - zwraca rezerwacje, które zostały opłacone po czasie 1 tygodnia

```
Create view BookingsPaidAfterDeadline as
Select CR.ConferenceReservationID, CR.ReservationDate, CR.PaymentDate,
DATEDIFF(day,CR.ReservationDate,CR.PaymentDate)-7 as DayDelay
From ConferenceReservations as CR
Where CR.PaymentDate is not NULL and
DATEDIFF(day,CR.ReservationDate,CR.PaymentDate)>7
Go
```

BookingsNotPaidYet - zwraca rezerwacje, które nie zostały jeszcze zapłacone, lecz nie minął jeszcze tydzień od rezerwacji

```
Create view BookingsNotPaidYet as
Select CR.ConferenceReservationID, CR.ReservationDate, CR.PaymentDate
From ConferenceReservations as CR
Where CR.PaymentDate is NULL and
DATEDIFF(day,CR.ReservationDate,getdate())<=7
Go
```

CorrectlyPaidBookings - zwraca rezerwacje, które zostały poprawnie opłacone

```
Create view CorrectlyPaidBookings as
Select CR.ConferenceReservationID, CR.ReservationDate, CR.PaymentDate
From ConferenceReservations as CR
Where CR.PaymentDate is not NULL and
DATEDIFF(day,CR.ReservationDate,CR.PaymentDate)<=7
Go
```

ClientStats - zwraca spis klientów wraz z informacją na temat liczby dni konferencji oraz warsztatów, w których uczestniczyli jak również kwotę, którą zapłacili za wszystkie te wydarzenia

```
CREATE view ClientStats as
Select C.ClientID, C.Name, C.Phone, C.Email, Ci.CityName, Co.CountryName,
(Select count(*) from ConferenceDayReservations as CDR JOIN ConferenceReservations as CR
on CDR.ConferenceReservationID=CR.ConferenceReservationID where CR.ClientID=C.ClientID) as TotalConferenceDays,
(Select count(*) from WorkshopReservations as WR JOIN ConferenceDayReservations as CDR
on WR.ConferenceDayReservationID=CDR.ConferenceDayReservationID JOIN ConferenceReservations as CR
on CDR.ConferenceReservationID=CR.ConferenceReservationID where CR.ClientID=C.ClientID) as TotalWorkshops,
((Select ISNULL(SUM(W.Price),0) from WorkshopReservations as WR JOIN ConferenceDayReservations as CDR
on WR.ConferenceDayReservationID=CDR.ConferenceDayReservationID JOIN ConferenceReservations as CR
on CDR.ConferenceReservationID=CR.ConferenceReservationID
JOIN Workshops as W on W.WorkshopID=WR.WorkshopID WHERE CR.ClientID=C.ClientID) +
(SELECT SUM(Co.Price*(1-ISNULL(discount,0))) FROM Conferences AS Co
JOIN ConferenceDays as CD on CD.ConferenceID=Co.ConferenceID
JOIN ConferenceDayReservations AS CDR ON CDR.ConferenceDayID = CD.ConferenceDayID
JOIN ConferenceReservations AS CR ON CR.ConferenceReservationID = CDR.ConferenceReservationID
LEFT JOIN Prices as P on P.ConferenceID=Co.ConferenceID and CR.PaymentDate between P.StartDate and P.EndDate
Where CR.ClientID=C.ClientID)) as TotalPaid
From Clients as C JOIN Cities as Ci on C.CityID=Ci.CityID JOIN Countries as Co on Co.CountryID=Ci.CountryID
Go
```

ConferencesAndWorkshops - zwraca spis konferencji wraz z dniami i warsztatami

```
CREATE view ConferencesAndWorkshops as
Select C.ConferenceID,C.ConferenceName,CD.ConferenceDayID,W.WorkshopID,WD.WorkshopName, C.StartDate AS ConferenceStart, C.EndDate AS
ConferenceEnd, W.StartDate AS WorkshopDate
from Conferences as C JOIN ConferenceDays as CD on C.ConferenceID=CD.ConferenceID
JOIN Workshops as W on W.ConferenceDayID=CD.ConferenceDayID
JOIN WorkshopDetails as WD on WD.WorkshopDetailsID=W.WorkshopDetailsID
Go
```

ConferenceReservationInfo - zwraca opis każdej rezerwacji wraz z informacjami dotyczącymi dni, warsztatów oraz ich cen

```
CREATE VIEW ConferenceReservationInfo AS
SELECT CR.ConferenceReservationID, CDR.ConferenceDayReservationID, WR.WorkshopReservationID, WR.WorkshopID,
((SELECT CDR.NumberOfParticipants*SUM(C.Price*(1-ISNULL(P.Discount,0))) FROM ConferenceDays AS CD JOIN Conferences AS C
ON C.ConferenceID = CD.ConferenceID LEFT JOIN dbo.Prices AS P
ON P.ConferenceID = C.ConferenceID AND CR.ReservationDate BETWEEN P.StartDate AND P.EndDate WHERE CD.ConferenceDayID=CDR.ConferenceDayID) +
(SELECT CDR.NumberOfStudents*SUM((1-ISNULL(C.StudentDiscount,0))*C.Price*(1-ISNULL(P.Discount,0))) FROM ConferenceDays AS CD JOIN Conferences AS C
ON C.ConferenceID = CD.ConferenceID LEFT JOIN dbo.Prices AS P
ON P.ConferenceID = C.ConferenceID AND CR.ReservationDate BETWEEN P.StartDate AND P.EndDate WHERE CD.ConferenceDayID=CDR.ConferenceDayID)) AS ConferenceDayPrice,
(SELECT WR.NumberOfParticipants*W.Price) AS WorkshopPrice
FROM ConferenceReservations AS CR JOIN ConferenceDayReservations AS CDR ON CR.ConferenceReservationID=CDR.ConferenceReservationID
JOIN WorkshopReservations AS WR ON WR.ConferenceDayReservationID=CDR.ConferenceDayReservationID
JOIN dbo.Workshops AS W ON W.WorkshopID = WR.WorkshopID
GO
```

Procedury

AddConference - dodaje konferencję do tabeli Conferences oraz odpowiednią ilość dni konferencji do tabeli ConferenceDays

```
Create Procedure AddConference
@Price money,
@ConferenceName varchar(30),
@ConferenceDescription varchar(150),
@StartDate datetime,
@EndDate datetime,
@ParticipantsLimit int,
@Address varchar(60)
As
Begin Try
    Insert into Conferences
    Values (@Price,@ConferenceName,@ConferenceDescription,@StartDate,@EndDate,NULL,
    @ParticipantsLimit,@Address)
    Declare @cnt int = 0;
    Declare @ConferenceID int = (Select C.ConferenceID from Conferences as C
    where C.ConferenceName=@ConferenceName);
    While @cnt <= DATEDIFF(day,@StartDate,@EndDate)
    Begin
        Insert into ConferenceDays
        Values (@ConferenceID)
        Set @cnt = @cnt + 1
    End;
End Try
Begin Catch
    Declare @errorMessage nvarchar(1024);
    Set @errorMessage = 'Error when adding conference: \n' + ERROR_MESSAGE();
    throw 52000, @errorMessage, 1;
End Catch
Go
```

AddPriceThreshold - dodaje próg cenowy do tabeli Prices

```
Create Procedure AddPriceThreshold
@ConferenceID int,
@Discount decimal(3,3),
@StartDate datetime,
@EndDate datetime
As
Begin TRY
    IF @EndDate < @StartDate
    Begin
        RAISERROR('The end date is before start!',16,1);
    End;
    DECLARE @start DATETIME = (SELECT StartDate FROM Conferences WHERE ConferenceID = @ConferenceID);
    IF @EndDate > @start
    Begin
        RAISERROR('The date range is post conference!',16,1);
    End;
    DECLARE @disc DECIMAL(3,3) = (SELECT MIN(Discount) FROM Prices WHERE ConferenceID=@ConferenceID
    AND EndDate<@EndDate);
    IF @disc < @Discount
    Begin
        RAISERROR('The discount is bigger than previous!',16,1);
    End;
    Insert into Prices
    Values (@ConferenceID,@Discount,@StartDate,@EndDate)
End Try
Begin Catch
    Declare @errorMessage nvarchar(1024);
    Set @errorMessage = 'Error when adding price threshold: \n' + ERROR_MESSAGE();
    throw 52000, @errorMessage, 1;
End Catch
Go
```

DeletePriceThreshold - usuwa próg cenowy z tabeli Prices

```
Create Procedure DeletePriceThreshold
@PriceID int
As
Begin Try
    Delete From Prices Where PriceID=@PriceID
End Try
Begin Catch
    Declare @errorMessage nvarchar(1024);
    Set @errorMessage = 'Error when deleting price threshold: \n' + ERROR_MESSAGE();
    throw 52000, @errorMessage, 1;
End Catch
Go
```

AddStudentDiscount - dodaje zniżkę studencką do danej konferencji

```
Create Procedure AddStudentDiscount
@ConferenceID int,
@studentDiscount decimal(3,3)
As
Begin Try
    Update Conferences
        Set StudentDiscount=@StudentDiscount
        Where ConferenceID = @ConferenceID
End Try
Begin Catch
    Declare @errorMessage nvarchar(1024);
    Set @errorMessage = 'Error when adding student discount: \n' + ERROR_MESSAGE();
    throw 52000, @errorMessage, 1;
End Catch
Go
```

AddWorkshop - dodaje warsztat do tabeli Workshops oraz informacje słownikowe na temat warsztatu do tabeli WorkshopDetails

```
Create Procedure AddWorkshop
@ConferenceDayID int,
@WorkshopName varchar(30),
@WorkshopDescription varchar(150),
@Price money,
@Address varchar(60),
@StartDate datetime,
@EndDate datetime,
@ParticipantsLimit int
As
Begin Try
    If (Select top 1 ParticipantsLimit from Conferences as C JOIN ConferenceDays as CD on C.ConferenceID=CD.ConferenceID
        Where CD.ConferenceID=@ConferenceDayID) < @ParticipantsLimit
    Begin
        RAISERROR('ParticipantsLimit bigger than in Conference!',16,1);
    End;
    Declare @ConferenceID int = (Select ConferenceID from ConferenceDays where ConferenceDayID=@ConferenceDayID);
    Declare @first_day int = (Select top 1 ConferenceDayID from ConferenceDays where ConferenceID = @ConferenceID);
    If (Select DATEPART(Day,C.StartDate) @first_day+@ConferenceDayID From Conferences as C
        Where ConferenceID=@ConferenceID) != DATEPART(Day,@StartDate) or DATEPART(Hour,@StartDate) >= DATEPART(Hour,@EndDate)
    Begin
        RAISERROR('The date is incorrect!',16,1);
    End;
    Insert into WorkshopDetails
    Values (@WorkshopName,@WorkshopDescription,@Price,@Address)
    Insert into Workshops
    Values (@ConferenceDayID,(Select max(WorkshopDetailsID) From WorkshopDetails),
        @Price,@StartDate,@EndDate,@ParticipantsLimit)
End Try
Begin Catch
    Declare @errorMessage nvarchar(1024);
    Set @errorMessage = 'Error when adding workshop: \n' + ERROR_MESSAGE();
    throw 52000, @errorMessage, 1;
End Catch
Go
```

ChangeConferenceInformation - zmienia informacje na temat danej konferencji

```

Create Procedure ChangeConferenceInformation
@ConferenceID int,
@Price money = NULL,
@ConferenceName varchar(30) = NULL,
@ConferenceDescription varchar(150) = NULL,
@StudentDiscount decimal(3,3) = NULL,
@Address varchar(60) = NULL
As
Begin Try
    Update Conferences
    Set Price=ISNULL(@Price,Price),
        ConferenceName=ISNULL(@ConferenceName,ConferenceName),
        ConferenceDescription=ISNULL(@ConferenceDescription,ConferenceDescription),
        StudentDiscount=ISNULL(@StudentDiscount,StudentDiscount),
        Address=ISNULL(@Address,Address)
    Where ConferenceID=@ConferenceID
End Try
Begin Catch
    Declare @errorMessage nvarchar(1024);
    Set @errorMessage = 'Error while changing conference information: \n' + ERROR_MESSAGE();
    throw 52000, @errorMessage, 1;
End Catch
Go

```

ChangeWorkshopInformation - zmienia informacje na temat danego warsztatu

```

Create Procedure ChangeWorkshopInformation
@WorkshopID int,
@Price money = NULL,
@WorkshopName varchar(30) = NULL,
@WorkshopDescription varchar(150) = NULL,
@Address varchar(60) = NULL
As
Begin Try
    Update Workshops
    Set Price=ISNULL(@Price,Price)
    Where WorkshopID=@WorkshopID
    Update WorkshopDetails
    Set WorkshopName=ISNULL(@WorkshopName,WorkshopName),
        WorkshopDescription=ISNULL(@WorkshopDescription,WorkshopDescription),
        Address=ISNULL(@Address,Address)
    Where WorkshopDetailsID=(Select WorkshopDetailsID from Workshops Where WorkshopID=@WorkshopID)
End Try
Begin Catch
    Declare @errorMessage nvarchar(1024);
    Set @errorMessage = 'Error while changing workshop information: \n' + ERROR_MESSAGE();
    throw 52000, @errorMessage, 1;
End Catch
Go

```

AddPrivateClient - dodaje klienta indywidualnego do tabeli Clients oraz tabel PrivateClients oraz Person

```

Create Procedure AddPrivateClient
@Name varchar(30),
@CityName varchar(30),
@CountryName varchar(30),
@Phone char(12),
@Email varchar(30)
As
Begin Try
    If (Select CountryID from Countries Where CountryName=@CountryName) is NULL
    Begin
        Insert into Countries Values (@CountryName)
    End;
    Declare @CountryID int = (Select CountryID from Countries Where CountryName=@CountryName);
    If (Select CityID from Cities Where CityName=@CityName) is NULL
    Begin
        Insert into Cities Values (@CityName,@CountryID)
    End;
    Declare @CityID int = (Select CityID from Cities Where CityName=@CityName);
    Insert into Clients
    Values (@Name,@CityID,@Phone,@Email)
End Try
Begin Catch
    Declare @errorMessage nvarchar(1024);
    Set @errorMessage = 'Error when adding private client: \n' + ERROR_MESSAGE();
    throw 52000, @errorMessage, 1;
End Catch
Go

```

UpdatePrivateClient - zmienia informacje danego klienta indywidualnego

```

Create Procedure UpdatePrivateClient
@clientID int,
@Name varchar(30) = NULL,
@cityID int = NULL,
@Phone char(12) = NULL,
@email varchar(30) = NULL
As
Begin Try
    Update Clients
        Set Name=ISNULL(@Name,Name),
        CityID=ISNULL(@CityID,CityID),
        Phone=ISNULL(@Phone,Phone),
        Email=ISNULL(@Email,Email)
        Where ClientID=@clientID
End Try
Begin Catch
    Declare @errorMessage nvarchar(1024);
    Set @errorMessage = 'Error while updating private client: \n' + ERROR_MESSAGE();
    throw 52000, @errorMessage, 1;
End Catch
Go

```

AddCompanyClient - dodaje klienta - firmę do tabeli Clients i tabeli Companies oraz odpowiednią ilość pracowników do tabeli CompanyEmployees oraz Person

```

Create Procedure AddCompanyClient
@Name varchar(30),
@cityName varchar(30),
@countryName varchar(30),
@Phone char(12),
@email varchar(30),
@CompanyName varchar(30),
@NIP char(10),
@ContactName varchar(30)
As
Begin Try
    If (Select CountryID from Countries Where CountryName=@CountryName) is NULL
        Begin
            Insert into Countries Values (@CountryName)
        End;
    Declare @CountryID int = (Select CountryID from Countries Where CountryName=@CountryName);
    If (Select CityID from Cities Where CityName=@CityName) is NULL
        Begin
            Insert into Cities Values (@CityName,@CountryID)
        End;
    Declare @CityID int = (Select CityID from Cities Where CityName=@CityName);
    Insert into Clients
        Values (@Name,@CityID,@Phone,@Email)
    Declare @ClientID int = (Select max(ClientID) from clients);
    Insert into Companies
        Values (@ClientID,@CompanyName,@NIP,@ContactName)
End Try
Begin Catch
    Declare @errorMessage nvarchar(1024);
    Set @errorMessage = 'Error when adding company client: \n' + ERROR_MESSAGE();
    throw 52000, @errorMessage, 1;
End Catch
Go

```

UpdateCompanyClient - zmienia informacje na temat danej firmy

```

Create Procedure UpdateCompanyClient
@clientID int,
@Name varchar(30) = NULL,
@CityID int = NULL,
@Phone char(12) = NULL,
@email varchar(30) = NULL,
@CompanyName varchar(30) = NULL,
@NIP char(10) = NULL,
@ContactName varchar(30) = NULL
As
Begin Try
    Update Clients
        Set Name=ISNULL(@Name,Name),
        CityID=ISNULL(@CityID,CityID),
        Phone=ISNULL(@Phone,Phone),
        Email=ISNULL(@Email,Email)
        Where ClientID=@clientID
    Update Companies
        Set CompanyName=ISNULL(@CompanyName,CompanyName),
        NIP=ISNULL(@NIP,NIP),
        ContactName=ISNULL(@ContactName,ContactName)
        Where CompanyID=@clientID
End Try
Begin Catch
    Declare @errorMessage nvarchar(1024);
    Set @errorMessage = 'Error while updating company client: \n' + ERROR_MESSAGE();
    throw 52000, @errorMessage, 1;
End Catch
Go

```

DeleteIncorrectReservations - usuwa wszystkie niepoprawne rezerwacje w momencie wywołania

```

Create Procedure DeleteIncorrectReservations
As
Begin Try
    Declare @cnt int = 1;
    Declare @size int = (Select count(*) from ConferenceReservations);
    While @cnt <= @size
    Begin
        Declare @res datetime = (Select ReservationDate from ConferenceReservations Where
        ConferenceReservationID=@cnt);
        Declare @pay datetime = (Select PaymentDate from ConferenceReservations Where
        ConferenceReservationID=@cnt);
        If DATEDIFF(DAY,@res,ISNULL(@pay,GETDATE()))>14
        Begin
            Delete from ConferenceReservations where ConferenceReservationID=@cnt
            Declare @days table (ConferenceDayReservationID INT NOT NULL)
            Insert into @days Select ConferenceDayReservationID from
            ConferenceDayReservations where ConferenceReservationID=@cnt
            Delete from ConferenceParticipants where ConferenceDayReservationID in
            (Select ConferenceDayReservationID from @days)
            Delete from WorkshopReservations where ConferenceDayReservationID in
            (Select ConferenceDayReservationID from @days)
            Declare @work_par table (WorkshopReservationID INT NOT NULL)
            Insert into @work_par Select WorkshopReservationID from WorkshopReservations
            where ConferenceDayReservationID in (Select ConferenceDayReservationID from @days)
            Delete from WorkshopParticipants where WorkshopReservationID in
            (Select WorkshopReservationID from @work_par)
            Delete from ConferenceDayReservations where ConferenceReservationID=@cnt
        End;
        Set @cnt = @cnt + 1
    End;
End Try
Begin Catch
    Declare @errorMessage nvarchar(1024);
    Set @errorMessage = 'Error while deleting incorrect reservations: \n' + ERROR_MESSAGE();
    throw 52000, @errorMessage, 1;
End Catch
Go

```

AddPayment - procedura uzupełniająca rezerwację o PaymentDate, wymagana poprawna cena rezerwacji (lub większa)

```

Create Procedure AddPayment
@ConferenceReservationID int,
@price int
As
Begin Try
    Declare @Res_price int = ((Select sum((1-ISNULL(Discount,0)*C.Price)) from ConferenceReservations as CR JOIN
    ConferenceDayReservations as CDR on CR.ConferenceReservationID=CDR.ConferenceReservationID JOIN ConferenceDays as CD on
    CDR.ConferenceDayID=CD.ConferenceDayID JOIN Conferences as C on C.ConferenceID=CD.ConferenceID
    JOIN Prices as P on (C.ConferenceID=P.ConferenceID and CR.ReservationDate between P.StartDate and P.EndDate)
    where CDR.ConferenceReservationID=@ConferenceReservationID) +
    (Select sum(W.Price) from ConferenceDayReservations as CDR JOIN WorkshopReservations as WR on
    WR.ConferenceDayReservationID=CDR.ConferenceDayReservationID JOIN Workshops as W on
    W.WorkshopID=WR.WorkshopID where CDR.ConferenceReservationID=@ConferenceReservationID));
    If @price >= @Res_price
    Begin
        Update ConferenceReservations
        Set PaymentDate = GETDATE()
        Where ConferenceReservationID=@ConferenceReservationID
    End;
End Try
Begin Catch
    Declare @errorMessage nvarchar(1024);
    Set @errorMessage = 'Error when adding payment: \n' + ERROR_MESSAGE();
    throw 52000, @errorMessage, 1;
End Catch
Go

```

AddConferenceBooking - dodaje rezerwację do ConferenceReservations

```

Create procedure AddConferenceBooking
@ClientID int
As
Begin Try
    Insert into ConferenceReservations
    Values (@ClientID, getdate(), NULL)
End Try
Begin Catch
    Declare @errorMessage nvarchar(1024);
    Set @errorMessage = 'Error when adding conference booking: \n' + ERROR_MESSAGE();
    throw 52000, @errorMessage, 1;
End Catch
Go

```

AddConferenceDayCompanyBooking - dodaje rezerwację firmy dnia konferencji dla danej rezerwacji

```
Create Type StudentIDList as table (ID int IDENTITY(1,1) Primary Key, StudentID int) Go

Create Procedure AddConferenceDayCompanyBooking
@ConferenceReservationID int,
@ConferenceDayID int,
@NumberOfParticipants int,
@NumberOfStudents int,
@studentIDList StudentIDList READONLY
As
Begin Try
    Declare @free_places int = (Select top 1 ParticipantsLimit from ConferenceDayReservations as CDR JOIN ConferenceDays as CD on
        CD.ConferenceDayID=CDR.ConferenceDayID JOIN Conferences as C on C.ConferenceID=CD.ConferenceID)-
        (Select sum(NumberOfParticipants+NumberOfStudents) from ConferenceDayReservations Where ConferenceDayID=@ConferenceDayID);
    If (@NumberOfParticipants+@NumberOfStudents) > @free_places
        Begin
            RAISERROR('There is no enough places to book',16,1);
        End
    If (Select count(*) from @studentIDList) != @NumberOfStudents
        Begin
            RAISERROR('The studentid count is incorrect!',16,1);
        End
    Insert into ConferenceDayReservations
    Values (@ConferenceReservationID,@ConferenceDayID,@NumberOfParticipants,
    @NumberOfStudents)
    Declare @cnt int = 0;
    Declare @clientID int = (Select ClientID from ConferenceReservations where ConferenceReservationID=@ConferenceReservationID);
    Declare @conferenceDayReservationID int = (Select max(ConferenceDayReservationID) from ConferenceDayReservations);
    While @cnt < @NumberOfParticipants:@NumberOfStudents
        Begin
            Insert into Person Values (NULL,NULL)
            Declare @personID int = (Select max(PersonID) from Person);
            Insert into CompanyEmployees Values (@personID,@clientID)
            Declare @studentID int = (Select StudentID from @studentIDList Where ID=@cnt+1);
            Insert into ConferenceParticipants Values(@personID,@studentID,@conferenceDayReservationID)
            Set @cnt = @cnt + 1
        End
    End Try
    Begin Catch
        Declare @errorMessage nvarchar(1024);
        Set @errorMessage = 'Error when adding conference day booking: \n' + ERROR_MESSAGE();
        throw 52000, @errorMessage, 1;
    End Catch
Go
```

AddConferenceDayIndividualBooking - dodaje rezerwację osoby prywatnej dnia konferencji dla danej rezerwacji

```
Create Procedure AddConferenceDayIndividualBooking
@ConferenceReservationID int,
@ConferenceDayID int,
@NumberOfParticipants int,
@NumberOfStudents int,
@studentID int = NULL
As
Begin Try
    Declare @free_places int = (Select top 1 ParticipantsLimit from ConferenceDayReservations as CDR JOIN ConferenceDays as CD on
        CD.ConferenceDayID=CDR.ConferenceDayID JOIN Conferences as C on C.ConferenceID=CD.ConferenceID)-
        (Select sum(NumberOfParticipants+NumberOfStudents) from ConferenceDayReservations Where ConferenceDayID=@ConferenceDayID);
    If (@NumberOfParticipants+@NumberOfStudents) != 1
        Begin
            RAISERROR('This is not an individual booking',16,1);
        End
    If (@NumberOfParticipants+@NumberOfStudents) > @free_places
        Begin
            RAISERROR('There is no enough places to book',16,1);
        End
    Insert into ConferenceDayReservations
    Values (@ConferenceReservationID,@ConferenceDayID,@NumberOfParticipants,
    @NumberOfStudents)
    Declare @clientID int = (Select ClientID from ConferenceReservations where ConferenceReservationID=@ConferenceReservationID);
    Declare @conferenceDayReservationID int = (Select max(ConferenceDayReservationID) from ConferenceDayReservations);
    Insert into Person Values (NULL,NULL)
    Declare @personID int = (Select max(PersonID) from Person);
    Insert into PrivateClients Values (@personID,@clientID)
    Insert into ConferenceParticipants Values(@personID,@studentID,@conferenceDayReservationID)
End Try
Begin Catch
    Declare @errorMessage nvarchar(1024);
    Set @errorMessage = 'Error when adding conference day booking: \n' + ERROR_MESSAGE();
    throw 52000, @errorMessage, 1;
End Catch
Go
```

DeleteConferenceDayCompanyReservation - usuwa rezerwację dnia firmy

```
CREATE PROCEDURE DeleteConferenceDayCompanyReservation
@ConferenceDayReservationID int
AS
Begin Try
    DECLARE @Participants TABLE (ParticipantID int);
    INSERT INTO @Participants (ParticipantID) SELECT ParticipantID FROM
    ConferenceParticipants WHERE ConferenceDayReservationID=@ConferenceDayReservationID
    DECLARE @Person TABLE (PersonID int);
    INSERT INTO @Person (PersonID) SELECT PersonID FROM
    Person WHERE PersonID IN (SELECT * FROM @Participants)
    DELETE FROM CompanyEmployees WHERE PersonID IN (SELECT * FROM @Person)
    DELETE FROM WorkshopParticipants WHERE ParticipantID IN (SELECT * FROM @Participants)
    DELETE FROM ConferenceParticipants WHERE ParticipantID IN (SELECT * FROM @Participants)
    DELETE FROM Person WHERE PersonID IN (SELECT * FROM @Person)
    DELETE FROM WorkshopReservations WHERE ConferenceDayReservationID=@ConferenceDayReservationID
    DELETE FROM ConferenceDayReservations WHERE ConferenceDayReservationID=@ConferenceDayReservationID
End Try
Begin Catch
    Declare @errorMessage nvarchar(1024);
    Set @errorMessage = 'Error when deleting conference day booking: \n' + ERROR_MESSAGE();
    throw 52000, @errorMessage, 1;
End CATCH
GO
```

DeleteConferenceCompanyReservation - usuwa rezerwację firmy

```
CREATE PROCEDURE DeleteConferenceCompanyReservation
@ConferenceReservationID int
AS
Begin Try
    DECLARE @Days TABLE (ID int IDENTITY(1,1) PRIMARY KEY , ConferenceDayReservationID int);
    INSERT INTO @Days (ConferenceDayReservationID) SELECT ConferenceDayReservationID FROM
    ConferenceDayReservations WHERE ConferenceReservationID = @ConferenceReservationID
    DECLARE @cnt int = 0;
    DECLARE @limit int = (SELECT COUNT(*) FROM @Days);
    WHILE @cnt < @limit
    BEGIN
        DECLARE @DayID int = (SELECT ConferenceDayReservationID FROM @Days WHERE ID=@cnt+1);
        EXEC DeleteConferenceDayCompanyReservation @DayID
    END
End Try
Begin Catch
    Declare @errorMessage nvarchar(1024);
    Set @errorMessage = 'Error when deleting conference booking: \n' + ERROR_MESSAGE();
    throw 52000, @errorMessage, 1;
End CATCH
Go
```

DeleteConferenceDayIndividualReservation - usuwa rezerwację dnia osoby prywatnej

```
CREATE PROCEDURE DeleteConferenceDayIndividualReservation
@ConferenceDayReservationID int
AS
Begin Try
    DECLARE @Participants TABLE (ParticipantID int);
    INSERT INTO @Participants (ParticipantID) SELECT ParticipantID FROM
    ConferenceParticipants WHERE ConferenceDayReservationID=@ConferenceDayReservationID
    DECLARE @Person TABLE (PersonID int);
    INSERT INTO @Person (PersonID) SELECT PersonID FROM
    Person WHERE PersonID IN (SELECT * FROM @Participants)
    DELETE FROM PrivateClients WHERE PersonID IN (SELECT * FROM @Person)
    DELETE FROM WorkshopParticipants WHERE ParticipantID IN (SELECT * FROM @Participants)
    DELETE FROM ConferenceParticipants WHERE ParticipantID IN (SELECT * FROM @Participants)
    DELETE FROM Person WHERE PersonID IN (SELECT * FROM @Person)
    DELETE FROM WorkshopReservations WHERE ConferenceDayReservationID=@ConferenceDayReservationID
    DELETE FROM ConferenceDayReservations WHERE ConferenceDayReservationID=@ConferenceDayReservationID
End Try
Begin Catch
    Declare @errorMessage nvarchar(1024);
    Set @errorMessage = 'Error when deleting conference day booking: \n' + ERROR_MESSAGE();
    throw 52000, @errorMessage, 1;
End CATCH
Go
```

DeleteConferenceIndividualReservation - usuwa rezerwację osoby prywatnej

```
CREATE PROCEDURE DeleteConferenceIndividualReservation
@ConferenceReservationID int
AS
Begin Try
    DECLARE @Days TABLE (ID int IDENTITY(1,1) PRIMARY KEY , ConferenceDayReservationID int);
    INSERT INTO @Days (ConferenceDayReservationID) SELECT ConferenceDayReservationID FROM
    ConferenceDayReservations WHERE ConferenceReservationID = @ConferenceReservationID
    DECLARE @cnt int = 0;
    DECLARE @limit int = (SELECT COUNT(*) FROM @Days);
    WHILE @cnt < @limit
    BEGIN
        DECLARE @DayID int = (SELECT ConferenceDayReservationID FROM @Days WHERE ID=@cnt+1);
        EXEC DeleteConferenceDayIndividualReservation @DayID
    END
End Try
Begin Catch
    Declare @errorMessage nvarchar(1024);
    Set @errorMessage = 'Error when deleting conference booking: \n' + ERROR_MESSAGE();
    throw 52000, @errorMessage, 1;
End CATCH
Go
```

AddWorkshopBooking - dodaje rezerwację warsztatu dla danej rezerwacji dnia konferencji

```
Create Procedure AddWorkshopBooking
@ConferenceDayReservationID int,
@WorkshopID int,
@NumberofParticipants int,
@ParticipantList ParticipantList READONLY
As
Begin Try
    Declare @free_places int = (Select ParticipantsLimit from Workshops where WorkshopID=@WorkshopID)-(Select sum(NumberOfParticipants) from WorkshopReservations where WorkshopID=@WorkshopID);
    If @NumberofParticipants > @free_places
    Begin
        RAISERROR('There is no enough places to book',1,16);
    End
    Insert into WorkshopReservations
    Values (@WorkshopID,@ConferenceDayReservationID, @NumberofParticipants)
    Declare @WorkshopReservationID int = (Select max(WorkshopReservationID) from WorkshopReservations);
    Declare @cnt int = 0;
    While @cnt < @NumberofParticipants
    Begin
        Declare @ParticipantID int = (Select ParticipantID from @Participantlist where ID=@cnt+1);
        Insert into WorkshopParticipants Values (@ParticipantID,@WorkshopReservationID)
        Set @cnt = @cnt + 1
    End
End Try
Begin Catch
    Declare @errorMessage nvarchar(1024);
    Set @errorMessage = 'Error when adding workshop booking: \n' + ERROR_MESSAGE();
    throw 52000, @errorMessage, 1;
End CATCH
Go
```

UpdatePerson - uzupełnia dane na temat osoby (imię i nazwisko)

```
Create Procedure UpdatePerson
@PersonID int,
@FirstName varchar(30),
@LastName varchar(30)
As
Begin Try
    Update Person
    Set FirstName=@FirstName,
    LastName=@LastName
    Where PersonID=@PersonID
End Try
Begin Catch
    Declare @errorMessage nvarchar(1024);
    Set @errorMessage = 'Error when updating person: \n' + ERROR_MESSAGE();
    throw 52000, @errorMessage, 1;
End CATCH
Go
```

Funkcje

AvailableConferenceDayNumberOfPlaces - Zwraca ilość wolnych miejsc na dany dzień konferencji

```
CREATE FUNCTION AvailableConferenceDayNumberOfPlaces
(
    @ConferenceDayID INT
)
Returns INT
AS
BEGIN
Return
(
    select (c.ParticipantsLimit - ISNULL((sum(cdr.NumberOfParticipants)+ sum(cdr.NumberOfStudents)),0))
    from Conferences as c
    inner join ConferenceDays as cd on cd.ConferenceID = c.ConferenceID
    left join ConferenceDayReservations as cdr on cdr.ConferenceDayID = cd.ConferenceDayID
    where cd.ConferenceDayID = @ConferenceDayID
    group by c.ParticipantsLimit
)
END
GO
```

AvailableWorkshopNumberOfPlaces - Zwraca ilość wolnych miejsc na dany warsztat

```
CREATE FUNCTION AvailableWorkshopNumberOfPlaces
(
    @WorkshopID INT
)
Returns INT
AS
BEGIN
Return
(
    select (w.ParticipantsLimit - sum(wr.NumberOfParticipants))
    from Workshops as w
    inner join WorkshopReservations as wr on wr.WorkshopID = w.WorkshopID
    where w.WorkshopID = @WorkshopID
    group by w.WorkshopID, w.ParticipantsLimit
)
END
GO
```

ConferenceDays - Zwraca dni danej konferencji

```

CREATE FUNCTION DaysOfConference
(
    @ConferenceID INT
)
Returns TABLE
AS
Return
(
    select cd.ConferenceDayID, c.ParticipantsLimit
    from Conferences as c
    inner join ConferenceDays as cd on cd.ConferenceID = c.ConferenceID
    where c.ConferenceID = @ConferenceID
)
GO

```

WorkshopStartDate - Zwraca datę rozpoczęcia danego warsztatu

```

CREATE FUNCTION WorkshopStartDate
(
    @WorkshopID INT
)
Returns datetime
AS
BEGIN
Return
(
    select w.StartDate
    from Workshops as w
    where w.WorkshopID = @WorkshopID
)
END
GO

```

WorkshopEndDate - Zwraca datę zakończenia warsztatu

```

CREATE FUNCTION WorkshopEndDate
(
    @WorkshopID INT
)
Returns datetime
AS
BEGIN
Return
(
    select w.EndDate
    from Workshops as w
    where w.WorkshopID = @WorkshopID
)
END
GO

```

ConferencePriceThresholds - Zwraca progi cenowe konferencji

```

CREATE FUNCTION ConferencePriceThresholds
(
    @ConferenceID INT
)
Returns table
AS
Return
(
    select p.PriceID, p.StartDate, p.EndDate, p.Discount
    from Prices as p
    inner join Conferences as c on c.ConferenceID = p.ConferenceID
    where c.ConferenceID = 2
)
GO

```

DoWorkshopsOverlap - Zwraca informacje czy dane dwa warsztaty nachodzą na siebie

```

CREATE FUNCTION DoWorkshopsOverlap
(
    @Workshop1ID INT,
    @Workshop2ID INT
)
Returns bit
AS
BEGIN
    DECLARE @Start_1 datetime = [dbo].[WorkshopStartDate] ( @Workshop1ID ) ;
    DECLARE @End_1 datetime = [dbo].[WorkshopEndDate] ( @Workshop1ID ) ;
    DECLARE @Start_2 datetime = [dbo].[WorkshopStartDate] ( @Workshop2ID ) ;
    DECLARE @End_2 datetime = [dbo].[WorkshopEndDate] ( @Workshop2ID ) ;

    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

```

BookingDaysCost - Zwraca cenę rezerwacji dni konferencji

```

CREATE FUNCTION BookingDaysCost
(
    @ConferenceReservationID INT
)
Returns money
AS
BEGIN
Return
(
    select SUM(c.Price*(1 - ISNULL(c.StudentDiscount,0))*cdr.NumberOfStudents + c.Price*cdr.NumberOfParticipants )
    from ConferenceDayReservations as cdr
    inner join ConferenceDays as cd on cd.ConferenceDayID = cdr.ConferenceDayID
    inner join Conferences as c on c.ConferenceID = cd.ConferenceID
    where cdr.ConferenceReservationID = @ConferenceReservationID
    group by cdr.ConferenceReservationID
)
END
GO

```

BookingWorkshopsCost - Zwraca cenę rezerwacji warsztatów

```
CREATE FUNCTION BookingWorkshopsCost
(
    @WorkshopReservationID INT
)
Returns money
AS
BEGIN
Return
(
    select wr.NumberOfParticipants*w.Price
    from WorkshopReservations as wr
    inner join Workshops as w on wr.WorkshopID = w.WorkshopID
    where wr.WorkshopReservationID = @WorkshopReservationID
)
END
GO
```

BookingCost - Zwraca pełną cenę rezerwacji

```
CREATE FUNCTION BookingCost
(
    @WorkshopReservationID INT,
    @ConferenceReservationID INT
)
Returns money
AS
BEGIN
DECLARE @BookingDaysCost money = dbo.BookingDaysCost ( @ConferenceReservationID );
DECLARE @BookingWorkshopsCost money = dbo.BookingWorkshopsCost ( @ConferenceReservationID );
RETURN (@BookingDaysCost + @BookingWorkshopsCost)
END
GO
```

ConferenceParticipantsList - Zwraca listę wszystkich uczestników danej konferencji

```
CREATE FUNCTION ConferenceParticipantsList
(
    @ConferenceID INT
)
Returns table
AS
Return
(
    select cp.ParticipantID, p.FirstName, p.LastName
    from Conferences as c
    inner join ConferenceDays as cd on cd.ConferenceID = c.ConferenceID
    inner join ConferenceDayReservations as cdr on cdr.ConferenceDayID = cd.ConferenceDayID
    inner join ConferenceParticipants as cp on cp.ConferenceDayReservationID = cdr.ConferenceDayReservationID
    inner join Person as p on p.PersonID = cp.PersonID
    where c.ConferenceID = @ConferenceID
)
GO
```

ConferenceDayParticipantsList - Zwraca listę wszystkich uczestników danego dnia konferencji

```
CREATE FUNCTION ConferenceDayParticipantsList
(
    @ConferenceDayID INT
)
Returns table
AS
Return
(
    select cp.ParticipantID, p.FirstName, p.LastName
    from ConferenceDays as cd
    inner join ConferenceDayReservations as cdr on cdr.ConferenceDayID = cd.ConferenceDayID
    inner join ConferenceParticipants as cp on cp.ConferenceDayReservationID = cdr.ConferenceDayReservationID
    inner join Person as p on p.PersonID = cp.PersonID
    where cd.ConferenceDayID = @ConferenceDayID
)
GO
```

WorkshopParticipantsList - Zwraca listę wszystkich uczestników danego warsztatu

```
CREATE FUNCTION WorkshopParticipantsList
(
    @WorkshopID INT
)
Returns table
AS
Return
(
    select cp.ParticipantID, p.FirstName, p.LastName
    from Workshops as w
    inner join WorkshopReservations as wr on wr.WorkshopID = w.WorkshopID
    inner join WorkshopParticipants as wp on wp.WorkshopReservationID = wr.WorkshopReservationID
    inner join ConferenceParticipants as cp on cp.ParticipantID = wp.ParticipantID
    inner join Person as p on p.PersonID = cp.PersonID
    where w.WorkshopID = @WorkshopID
)
GO
```

ParticipantsConferencesList - Zwraca listę wszystkich konferencji, w których dana osoba uczestniczyła lub uczestniczy

```
CREATE FUNCTION ParticipantsConferencesList
(
    @ParticipantID INT
)
Returns table
AS
Return
(
    select c.ConferenceID, c.ConferenceName
    from ConferenceParticipants as cp
    inner join ConferenceDayReservations as cdr on cdr.ConferenceDayReservationID = cp.ConferenceDayReservationID
    inner join ConferenceDays as cd on cd.ConferenceDayID = cdr.ConferenceDayID
    inner join Conferences as c on c.ConferenceID = cd.ConferenceID
    where cp.ParticipantID = @ParticipantID
    group by c.ConferenceID, c.ConferenceName
)
GO
```

ParticipantsWorkshopsList - Zwraca listę wszystkich warsztatów, w których dana osoba uczestniczyła lub uczestniczy

```
CREATE FUNCTION ParticipantsWorkshopsList
(
    @ParticipantID INT
)
Returns table
AS
Return
(
    select w.WorkshopID, wd.WorkshopName
    from ConferenceParticipants as cp
    inner join WorkshopParticipants as wp on wp.ParticipantID = cp.ParticipantID
    inner join WorkshopReservations as wr on wr.WorkshopReservationID = wp.WorkshopReservationID
    inner join Workshops as w on w.WorkshopID = wr.WorkshopID
    inner join WorkshopDetails as wd on wd.WorkshopDetailsID = w.WorkshopDetailsID
    where cp.ParticipantID = @ParticipantID
    group by w.WorkshopID, wd.WorkshopName
)
GO
```

TotalCostOfReservation - zwraca cały koszt rezerwacji

```
CREATE FUNCTION TotalCostOfReservation
(
    @ConferenceReservationID INT
)
Returns money
AS
BEGIN
    declare @WorkshopCost money = (select sum(w.Price*wr.NumberOfParticipants)
    from ConferenceDayReservations as cdr
    inner join WorkshopReservations as wr on wr.ConferenceDayReservationID = cdr.ConferenceDayReservationID
    inner join Workshops as w on w.WorkshopID = wr.WorkshopID
    where cdr.ConferenceReservationID = @ConferenceReservationID)

    declare @ConferenceCost money = (select SUM(c.Price*(1 - ISNULL(c.StudentDiscount,0))*cdr.NumberOfStudents + c.Price*cdr.NumberOfParticipants )
        from ConferenceDayReservations as cdr
        inner join ConferenceDays as cd on cd.ConferenceDayID = cdr.ConferenceDayID
        inner join Conferences as c on c.ConferenceID = cd.ConferenceID
        where cdr.ConferenceReservationID = @ConferenceReservationID)

    Return
    (
        (@WorkshopCost + @ConferenceCost)
    )
END
GO
```

Triggery

TooFewFreePlacesForConferenceDay - sprawdza, czy ilość miejsc dostępnych danego dnia konferencji nie jest mniejsza od ilości miejsc zarezerwowanych w tworzonej rezerwacji.

```

CREATE TRIGGER [dbo].[TooFewFreePlacesForConferenceDay]
on [dbo].[ConferenceDayReservations]
AFTER INSERT
AS
BEGIN
SET NOCOUNT ON;
IF EXISTS
(
SELECT * FROM inserted AS i
WHERE dbo.AvailableConferenceDayNumberOfPlaces(i.ConferenceDayID) < 0
)
BEGIN
; THROW 50001 , 'Too few free places to book day' ,1
END
END
GO

```

TooFewFreePlacesForWorkshop - sprawdza, czy ilość miejsc dostępnych danego warsztatu nie jest mniejsza od ilości miejsc zarezerwowanych w tworzonej rezerwacji.

```

CREATE TRIGGER [dbo].[TooFewFreePlacesForWorkshop]
on [dbo].[WorkshopReservations]
AFTER INSERT
AS
BEGIN
SET NOCOUNT ON;
IF EXISTS
(
SELECT * FROM inserted AS i
WHERE dbo.AvailableWorkshopNumberOfPlaces(i.WorkshopID) < 0
)
BEGIN
; THROW 50001 , 'Too few free places to book Workshop' ,1
END
END
GO

```

LessPlacesBookedForDayThanForWorkshop - Blokuje rezerwację na warsztat, jeżeli klient zarezerwował mniej miejsc na dzień niż warsztat

```

CREATE TRIGGER [dbo].[LessPlacesBookedForDayThanForWorkshop]
ON [dbo].[WorkshopReservations]
AFTER INSERT
AS
BEGIN

SET NOCOUNT ON;
IF EXISTS
(
SELECT * FROM inserted AS i
JOIN ConferenceDayReservations AS cdr ON cdr.ConferenceDayReservationID = i.ConferenceDayReservationID
WHERE cdr.NumberOfParticipants + cdr.NumberOfStudents
< i.NumberOfParticipants
)
BEGIN
; THROW 50001 , 'Cannot book Workshop since your Conference Day Reservation has less places booked' ,1
END
END
GO

```

ConferenceDayHasLessPlacesThanWorkshop - blokuje możliwość dodania warsztatu, jeżeli liczba miejsc warsztatu jest większa niż na dany dzień konferencji.

```
CREATE TRIGGER [dbo].[ConferenceDayHasLessPlacesThanWorkshop]
ON [dbo].[Workshops]
AFTER INSERT
AS
BEGIN
SET NOCOUNT ON;
IF EXISTS
(
    select *
    from inserted as i
    join ConferenceDays as cd on cd.ConferenceDayID = i.ConferenceDayID
    join Conferences as c on c.ConferenceID = cd.ConferenceID
    where i.ParticipantsLimit > c.ParticipantsLimit
)
BEGIN
; THROW 50001 , 'You can not create Workshop with more places than for a Conference Day' ,1
END
END
GO
```

BookingWorkshopInDifferentDay - blokuje możliwość zarezerwowania warsztatu na inny dzień niż się zrobiło rezerwacje na ConfDay

```
CREATE TRIGGER [dbo].[BookingWorkshopInDifferentDay]
ON [dbo].[WorkshopReservations]
AFTER INSERT
AS
BEGIN
SET NOCOUNT ON;
IF EXISTS
(
    select *
    from inserted as i
    join Workshops as w on w.WorkshopID = i.WorkshopID
    join ConferenceDays as cd1 on cd1.ConferenceDayID = w.ConferenceDayID
    join ConferenceDayReservations as cdr on cdr.ConferenceDayReservationID = i.ConferenceDayReservationID
    join ConferenceDays as cd2 on cd2.ConferenceDayID = cdr.ConferenceDayID
    where cd1.ConferenceDayID <> cd2.ConferenceDayID
)
BEGIN
; THROW 50001 , 'Booking workshop from different day than day booking is for' ,1
END
END
GO
```

ParticipantReservatingOverlappingWorkshop - Blokuje możliwość zapisania się uczestnika na warsztat pokrywający się z warsztatem, na który uczestnik już się zapisał.

```
CREATE TRIGGER ParticipantReservatingOverlappingWorkshop
ON [dbo].[WorkshopReservations]
AFTER INSERT
AS
BEGIN
SET NOCOUNT ON;
IF EXISTS
(
SELECT * FROM inserted AS i
inner join WorkshopParticipants as wp on wp.WorkshopReservationID = i.WorkshopReservationID
cross apply dbo.ParticipantsWorkshopsList(wp.ParticipantID) as w
where dbo.DoWorkshopsOverlap (i.WorkshopID,w.WorkshopID) = 1 and w.WorkshopID <> i.WorkshopID
)
BEGIN
; THROW 50001 , 'Workshops you are trying to book overlap' ,1
END
END
GO
```

Indeksy

idx_Countries - dodaje indeks do tabeli Countries dla kolumny CountryName

```
CREATE INDEX idx_Countries
ON Countries (CountryName)
GO
```

idx_Cities - dodaje indeks do tabeli Cities dla kolumny CityName

```
CREATE INDEX idx_Cities
ON Cities (CityName)
GO
```

idx_Clients - dodaje indeks do tabeli Clients dla kolumny ClientID

```
|CREATE INDEX idx_Clients
ON Clients (ClientID)
GO
```

idx_Person - dodaje indeksy do tabeli Person dla kolumn FirstName, LastName

```
CREATE INDEX idx_Person  
ON Person (FirstName, LastName)  
GO
```

idx_ConferenceReservations - dodaje indeks do tabeli ConferenceReservations dla kolumny ConferenceReservationID

```
CREATE INDEX idx_ConferenceReservations  
ON ConferenceReservations (ConferenceReservationID)  
GO
```

idx_ConferenceDayReservations - dodaje indeks do tabeli ConferenceDayReservations dla kolumny ConferenceDayReservationID

```
CREATE INDEX idx_ConferenceDayReservations  
ON ConferenceDayReservations (ConferenceDayReservationID)  
GO
```

idx_ConferenceParticipants - dodaje indeks do tabeli ConferenceParticipants dla kolumny ParticipantID

```
CREATE INDEX idx_ConferenceParticipants  
ON ConferenceParticipants (ParticipantID)  
GO
```

idx_WorkshopReservations - dodaje indeks do tabeli WorkshopReservations dla kolumny WorkshopReservationID

```
CREATE INDEX idx_WorkshopReservations  
ON WorkshopReservations (WorkshopReservationID)  
GO
```

idx_ConferenceDays - dodaje indeksy do tabeli ConferenceDays dla kolumn ConferenceDayID, ConferenceID

```
CREATE INDEX idx_ConferenceDays  
ON ConferenceDays (ConferenceDayID, ConferenceID)  
GO
```

idx_Workshops - dodaje indeks do tabeli Workshops dla kolumny WorkshopID

```
CREATE INDEX idx_Workshops  
ON Workshops (WorkshopID)  
GO
```

idx_Conferences - dodaje indeks do tabeli Conferences dla kolumny ConferenceID

```
CREATE INDEX idx_Conferences  
ON Conferences (ConferenceID)  
GO
```

idx_WorkshopDetails - dodaje indeks do tabeli WorkshopDetails dla kolumny WorkshopDetailsID

```
CREATE INDEX idx_WorkshopDetails  
ON WorkshopDetails (WorkshopDetailsID)  
GO
```

idx_Prices - dodaje indeksy do tabeli Prices dla kolumn PriceID, ConferenceID

```
|CREATE INDEX idx_Prices  
ON Prices (PriceID,ConferenceID)  
GO
```

Generator

Do wygenerowania danych użyliśmy generatora danych SQL firmy RedGate, która udostępnia swoje usługi pod stroną: <https://www.red-gate.com>.

Dane tabel, których wygenerowanie wymagało jedynie wygenerowania losowych wartości wygenerowaliśmy dostępnymi algorytmami programu SQL Data Generator. Tabele, które wymagały zachowania warunków integralności utworzyliśmy przy pomocy poniższych skryptów:

→ skrypt tworzący dni konferencji

```
DECLARE @ConferenceID INT = 80;  
|WHILE @ConferenceID <= (SELECT COUNT(*) FROM Conferences)  
|BEGIN  
    DECLARE @sdate DATETIME = (SELECT StartDate FROM Conferences WHERE @ConferenceID=ConferenceID);  
    DECLARE @edate DATETIME = (SELECT EndDate FROM Conferences WHERE @ConferenceID=ConferenceID);  
    DECLARE @days INT = (SELECT DATEDIFF(DAY,@sdate,@edate)) + 1;  
    DECLARE @cnt INT = 0;  
    WHILE @cnt < @days  
    BEGIN  
        INSERT INTO dbo.ConferenceDays VALUES (@ConferenceID)  
        SET @cnt = @cnt + 1  
    END  
    SET @ConferenceID = @ConferenceID + 1  
END
```

→ skrypt naprawiający daty warsztatów

```
DECLARE @WorkshopID INT = 1;  
|WHILE @WorkshopID <= (SELECT COUNT(*) FROM dbo.Workshops)  
|BEGIN  
    DECLARE @ConferenceDayID INT = (SELECT ConferenceDayID FROM dbo.Workshops WHERE @WorkshopID=WorkshopID);  
    DECLARE @ConferenceID INT = (SELECT ConferenceID FROM dbo.ConferenceDays WHERE ConferenceDayID=@ConferenceDayID);  
    DECLARE @firstday DATETIME = (SELECT StartDate FROM Conferences WHERE ConferenceID=@ConferenceID);  
    DECLARE @FirstConferenceDayID INT = (SELECT TOP 1 ConferenceDayID FROM dbo.ConferenceDays WHERE ConferenceID=@ConferenceID ORDER BY 1);  
    DECLARE @DayDiff INT = @ConferenceDayID - @FirstConferenceDayID;  
    DECLARE @RightDate DATETIME = DATEADD(DAY,@DayDiff,@firstday);  
    SET @RightDate = DATEADD(HOUR,(SELECT DATEPART(HOUR,(SELECT StartDate FROM dbo.Workshops WHERE WorkshopID=@WorkshopID)))-  
        (SELECT DATEPART(HOUR,@RightDate)),@RightDate);  
    DECLARE @EndRightDate DATETIME = DATEADD(HOUR,2,@RightDate);  
    UPDATE dbo.Workshops  
    SET StartDate = @RightDate, EndDate = @EndRightDate  
    WHERE WorkshopID = @WorkshopID;  
    SET @WorkshopID = @WorkshopID + 1  
END
```

→ skrypt dodający uczestników konferencji

```
DECLARE @ConferenceDayReservationID INT = 1;
DECLARE @PersonID INT = 1;
WHILE @ConferenceDayReservationID <= (SELECT COUNT(*) FROM dbo.ConferenceDayReservations)
BEGIN
    DECLARE @NumberOfParticipants INT = (SELECT NumberOfParticipants FROM dbo.ConferenceDayReservations
        WHERE @ConferenceDayReservationID=ConferenceDayReservationID);
    DECLARE @NumberOfStudents INT = (SELECT NumberOfStudents FROM dbo.ConferenceDayReservations
        WHERE @ConferenceDayReservationID=ConferenceDayReservationID);
    DECLARE @cnt INT = 0;
    WHILE @cnt < @NumberOfParticipants
    BEGIN
        INSERT INTO ConferenceParticipants VALUES (@PersonID,NULL,@ConferenceDayReservationID)
        SET @PersonID = @PersonID + 1
        SET @cnt = @cnt + 1
    END
    SET @cnt = 0;
    WHILE @cnt < @NumberOfStudents
    BEGIN
        INSERT INTO ConferenceParticipants VALUES (@PersonID,(SELECT FLOOR(RAND()*(999999-100000))+100000),@ConferenceDayReservationID)
        SET @PersonID = @PersonID + 1
        SET @cnt = @cnt + 1
    END
    SET @ConferenceDayReservationID = @ConferenceDayReservationID + 1;
END
```

→ skrypt dodający klientów firmowych

```
DECLARE @CompClients TABLE (ID INT IDENTITY(1,1) PRIMARY KEY, CompanyID INT );
INSERT INTO @CompClients SELECT CompanyID FROM Companies
DECLARE @ID INT = 1;
DECLARE @PersonID INT = 1;
WHILE @ID <= (SELECT COUNT(*) FROM Companies)
BEGIN
    DECLARE @CompanyID INT = (SELECT CompanyID FROM @CompClients WHERE ID=@ID);
    DECLARE @Number INT = (SELECT SUM(CDR.NumberOfParticipants+CDR.NumberOfStudents) FROM dbo.Companies AS Co
        JOIN dbo.Clients AS C ON C.ClientID = Co.CompanyID
        JOIN dbo.ConferenceReservations AS CR ON CR.ClientID = C.ClientID
        JOIN dbo.ConferenceDayReservations AS CDR ON CDR.ConferenceReservationID = CR.ConferenceReservationID
        WHERE C.ClientID=@CompanyID);
    DECLARE @cnt INT = 0;
    WHILE @cnt < @Number
    BEGIN
        INSERT INTO CompanyEmployees VALUES (@PersonID,@CompanyID)
        SET @PersonID = @PersonID + 1
        SET @cnt = @cnt + 1
    END
    SET @ID = @ID + 1;
END
```

→ skrypt dodający klientów indywidualnych

```
DECLARE @IndClients TABLE (ID INT IDENTITY(1,1) PRIMARY KEY, ClientID INT );
INSERT INTO @IndClients SELECT ClientID FROM Clients INTERSECT SELECT CompanyID FROM Companies
DECLARE @ID INT = 1;
DECLARE @PersonID INT = 19191;
WHILE @ID <= (SELECT COUNT(*) FROM @IndClients)
BEGIN
    DECLARE @ClientID INT = (SELECT ClientID FROM @IndClients WHERE ID=@ID);
    DECLARE @Number INT = (SELECT SUM(CDR.NumberOfParticipants+CDr.NumberOfStudents) from dbo.Clients AS C
    JOIN dbo.ConferenceReservations AS CR ON CR.ClientID = C.ClientID
    JOIN dbo.ConferenceDayReservations AS CDR ON CDR.ConferenceReservationID = CR.ConferenceReservationID
    WHERE C.ClientID=@ClientID);
    DECLARE @cnt INT = 0;
    WHILE @cnt < @Number
    BEGIN
        INSERT INTO PrivateClients VALUES (@PersonID,@ClientID)
        SET @PersonID = @PersonID + 1
        SET @cnt = @cnt + 1
    END
    SET @ID = @ID + 1;
END
```

→ skrypt dodający uczestników warsztatów

```
DECLARE @WorkshopReservationID INT = 1;
DECLARE @DayParticipantID INT = 1;
WHILE @WorkshopReservationID <= (SELECT COUNT(*) FROM dbo.WorkshopReservations)
BEGIN
    DECLARE @ConferenceDayReservationID INT = (SELECT ConferenceDayReservationID FROM dbo.WorkshopReservations
    WHERE WorkshopReservationID=@WorkshopReservationID);
    DECLARE @DayParticipants TABLE (ID INT IDENTITY(1,1) PRIMARY KEY, ParticipantID INT);
    INSERT INTO @DayParticipants SELECT ParticipantID FROM dbo.ConferenceDayReservations AS CDR
    JOIN dbo.ConferenceParticipants AS CP ON CP.ConferenceDayReservationID = CDR.ConferenceDayReservationID
    WHERE CDR.ConferenceDayReservationID=@ConferenceDayReservationID;
    DECLARE @Number INT = (SELECT NumberOfParticipants FROM dbo.WorkshopReservations
    WHERE WorkshopReservationID=@WorkshopReservationID);
    DECLARE @cnt INT = 0;
    WHILE @cnt < @Number
    BEGIN
        DECLARE @ParticipantID INT = (SELECT ParticipantID FROM @DayParticipants WHERE ID=@DayParticipantID);
        INSERT INTO WorkshopParticipants VALUES (@ParticipantID,@WorkshopReservationID)
        SET @DayParticipantID = @DayParticipantID + 1
        SET @cnt = @cnt + 1
    END
    SET @WorkshopReservationID = @WorkshopReservationID + 1
END
```

→ skrypt usuwający niepoprawne progi cenowe

```
DECLARE @PriceID INT = 1;
DECLARE @Count INT = (SELECT COUNT(*) FROM Prices)
WHILE @PriceID <= @Count
BEGIN
    DECLARE @CDate DATETIME = (SELECT C.StartDate FROM Conferences AS C JOIN dbo.Prices AS P
    ON P.ConferenceID = C.ConferenceID WHERE @PriceID=PriceID);
    DECLARE @EDate DATETIME = (SELECT EndDate FROM Prices WHERE @PriceID=PriceID);
    IF @CDate < @EDate
    BEGIN
        DELETE FROM Prices WHERE PriceID=@PriceID
    END
    SET @PriceID = @PriceID + 1
END
```

Liczba wierszy wygenerowanych dla każdej tabeli wyniosła:

Countries - 20

Clients - 100

Companies - 500

CompanyEmployees - 19190

ConferenceDayReservations - 3706

ConferenceDays - 307

ConferenceParticipants - 37056

ConferenceReservations - 1920

Conferences - 80

Person - 37056

Prices - 298

PrivateClients - 17866

WorkshopDetails - 1228

WorkshopParticipants - 44397

WorkshopReservations - 14824

Workshops - 1228