Politechnika Krakowska

im. Tadeusza Kościuszki

wydział WIEiK, kierunek Informatyka

rok III, semestr V

przedmiot: Systemy baz danych

SPRAWOZDANIE Z PROJEKTU

RAPORT III

OPRACOWANE PRZEZ:

PAWEŁ IRZYK

HUBERT KOPEĆ

Cel dokumentu

W dokumencie zaprezentowane zostaną: ogólny opis projektu, architektura i model danych składających się na projekt.

Cel projektu

Celem projektu jest stworzenie systemu do wspomagania działaności firmy organizującej konferencje.

Zakres projektu

1. Opis

Opis problemu

Projekt dotyczy systemu wspomagania działalności firmy organizującej konferencje

Ogólne informacje

Firma organizuje konferencje, które mogą być jedno- lub kilkudniowe. Klienci powinni móc rejestrować się na konferencje za pomocą systemu www. Klientami mogą być zarówno indywidualne osoby jak i firmy, natomiast uczestnikami konferencji są osoby (firma nie musi podawać od razu przy rejestracji listy uczestników - może zarezerwować odpowiednią ilość miejsc na określone dni oraz na warsztaty, natomiast na 2 tygodnie przed rozpoczęciem musi te dane uzupełnić - a jeśli sama nie uzupełni do tego czasu, to pracownicy dzwonią do firmy i ustalają takie informacje). Każdy uczestnik konferencji otrzymuje identyfikator imienny (+ ew. informacja o firmie na nim). Dla konferencji kilkudniowych, uczestnicy mogą rejestrować się na dowolne z tych dni.

Warsztaty

Ponadto z konferencją związane są warsztaty, na które uczestnicy także mogą się zarejestrować - muszą być jednak zarejestrowani tego dnia na konferencję, aby móc w nich uczestniczyć. Kilka warsztatów może trwać równocześnie, ale uczestnik nie może zarejestrować się na więcej niż jeden warsztat, który trwa w tym samym czasie. Jest także ograniczona ilość miejsc na każdy warsztat i na każdy dzień konferencji. Część warsztatów może być płatna, a część jest darmowa.

Opłaty

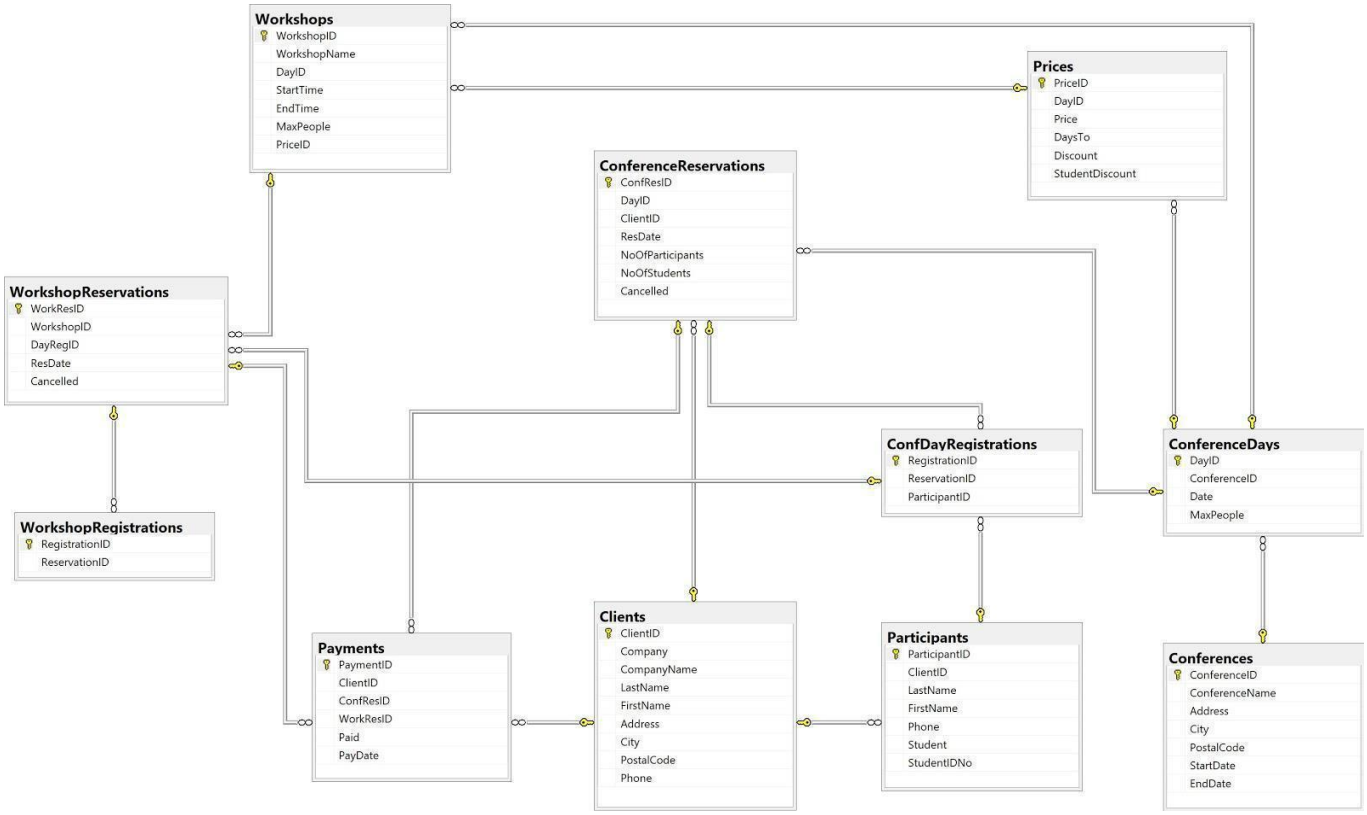
Opłata za udział w konferencji zależy nie tylko od zarezerwowanych usług, ale także od terminu ich rezerwacji - jest kilka progów ceny (progi ceny dotyczą tylko udziału w konferencji, cena warsztatów jest stała) i im bliżej rozpoczęcia konferencji, tym cena jest wyższa (jest także zniżka procentowa dla studentów i w takim wypadku przy rezerwacji trzeba podać nr legitymacji studenckiej). Na zapłatę klienci mają tydzień od rezerwacji na konferencję - jeśli do tego czasu nie pojawi się opłata, rezerwacja jest anulowana.

Raporty

Dla organizatora najbardziej istotne są listy osobowe uczestników na każdy dzień konferencji i na każdy warsztat, a także informacje o płatnościach klientów. Ponadto organizator chciałby mieć informację o klientach, którzy najczęściej korzystają z jego usług. Specyfika firmy Firma organizuje średnio 2 konferencje w miesiącu, każda z nich trwa zwykle 2-3 dni, w tym średnio w każdym dniu są 4 warsztaty. Na każdą konferencję średnio rejestruje się 200 osób. Stworzona baza danych powinna zostać wypełniona w takim stopniu, aby odpowiadała 3-letniej działalności firmy

---

1. Diagram bazy danych



2. Proces projektowania

Pierwszym krokiem w rozpoczęciu procesu projektowania było zapoznanie się jak wygląda praca firmy która odpowiedzialna jest za organizowanie konferenicji. Wiedza oraz uwagi o procesie rejestracji oraz donacji pomogły w modelowaniu systemu.
Dzięki spotkaniu z osobami odpowiedzialnymi za wdrożenie systemu, wiedzieliśmy jakie są oczekiwania. Na podstawie rozmów, można było ustalić wszystkie najistotniejsze parametry systemu oraz jego funkcje. Po rozmowie z przedstawicielami firmy, która organizuje konferencje wiedzieliśmy jakie są oczekiwania wobec systemu, którego stworzenie zostało nam powierzone. Znając oczekiwania, mogliśmy przemyśleć w jaki sposób wymagania klienta przełożyć na rzeczywistą aplikację. Po przeanalizowaniu uzyskanych informacji mogliśmy przystąpić do stworzenia systemu.

3. Normalizacja bazy danych

- pierwsza postać normalna 1NF

Pierwsza postać normalna mówi o atomowości danych. Czyli tabela (encja) przechowuje dane w
sposób atomowy. Każde pole przechowuje jedną informację, dzięki czemu możemy dokonywać efektywnych zapytań. Wprowadza także pojęcie istnienie klucza głównego identyfikującego bezpośrednio każdy wiersz –unikalności. Nasza baza danych znajduję się w 1NF, ponieważ każdy wiersz przechowuje informacje o pojedynczym obiekcie, nie zawiera kolekcji, encje posiadają klucz główny (kolumnę lub grupę kolumn jednoznacznie identyfikujących go w zbiorze) a dane są atomowe.

- Druga postać normalna 2NF

Mówi o tym, że każda tabela powinna przechowywać dane dotyczące tylko konkretnej klasy obiektów. W naszym diagramie ERD wszystkie zbiory atrybutów są zależne od swojego klucza głównego.

- Trzecia postać normalna 3NF

Trzecia postać normalna głosi, że kolumna informacyjna nie należąca do klucza nie zależy też od innej kolumny informacyjnej, nie należącej do klucza. W naszej sytuacji każdy niekluczowy argument jest bezpośrednio zależny tylko od klucza głównego, a nie od innej kolumny, dlatego baza danych spełnia warunki również dla trzeciej postaci normalnej.

4. Tabele

Conferences – Tabela przechowuje infrormacje o konferencjach. Zawiera identyfikator (klucz główny), nazwę konferencji, miejsce, w którym odbywa się oraz datę rozpoczęcia i zakończenia. Sprawdzana jest poprawność wprowadzonych dat, tak aby niemożliwe było wpisanie konferencji kończącej się wcześniej niż data jej rozpoczęcia.

```
CREATE TABLE Conferences (
      ConferenceID int not null primary key identity (1,1),
      ConferenceName nvarchar(60) not null,
      Address nvarchar(50) not null,
      City nvarchar(40) not null,
      PostalCode nvarchar (10) not null,
      StartDate date not null,
      EndDate date not null ,
)

ALTER TABLE Conferences WITH CHECK ADD CONSTRAINT [StartDateBeforeEndDate]
CHECK ((StartDate<=EndDate))
```

ConferenceDays – Tabela przechowuje informacje o poszczególnych dniach konferencji. Posiada identyfikator dnia (klucz główny), identyfikator konferencji odwołujący się do klucza głównego tabeli Conferences oraz datę i maksymalną liczbę osób mogących wziąć udział w danym dniu konferencji (nie można podać ujemnej).

```
CREATE TABLE ConferenceDays (
      DayID int not null primary key identity(1,1),
      ConferenceID int not null foreign key references Conferences(ConferenceID),
      Date date not null,
      MaxPeople int not null CHECK (MaxPeople>=0),
)
```

Prices – Tabela przechowuje informacje o progach cenowych dla poszczególnych dni konferencji. Zawiera identyfikator ceny (klucz główny), identyfikator dnia, dla którego dany próg obowiązuje, cenę oraz liczbę oznaczającą do ilu dni przed konferencją można zapłacić za uczestnika tę cenę. Oprócz tego w tabeli zawarte są informacje o zniżce dla określonej wartości DaysTo oraz o zniżce dla studentów na ten dzień. Zarówno każda ze zniżek, jak i ich suma nie może być większa od 1.

```
CREATE TABLE Prices (
	PriceID int not null primary key identity(1,1),
	DayID int null foreign key references ConferenceDays(DayID),
	Price numeric(2,2) not null CHECK (Price>0)
	DaysTo int not null CHECK (DaysTo>0),
	Discount numeric(2,2) not null DEFAULT 0 CHECK (DISCOUNT BETWEEN 0 AND 1),
	StudentDiscount numeric(2,2) not null DEFAULT 0 CHECK (StudentDiscount BETWEEN 0
		AND 1),
)

ALTER TABLE Prices WITH CHECK ADD CONSTRAINT [SumDiscountsBetween0And1]
CHECK ((Discount+StudentDiscount<=1))
```

Workshops – Tabela przechowuje informacje o warsztatach odbywających się podczas dni konferencji. Każdy z warsztatów ma identyfikator (klucz główny), nazwę, dzień, w którym się odbywa, czas rozpoczęcia i zakończenia, maksymalną liczbę uczestników oraz identyfikator ceny będący kluczem obcym. Podobnie jak w przypadku konferencji sprawdzane jest czy wprowadzony czas zakończenia jest późniejszy niż czas rozpoczęcia.

```
CREATE TABLE Workshops (
	WorkshopID int not null primary key identity(1,1),
	WorkshopName nvarchar (60) not null,
	DayID int not null foreign key references ConferenceDays(DayID),
	StartTime time(0) not null,
	EndTime time(0) not null,
	MaxPeople int not null CHECK (MaxPeople>=0),
	PriceID int null foreign key references Prices(PriceID),
)

ALTER TABLE Workshops WITH CHECK ADD CONSTRAINT [StartTimeBeforeEndTime]
CHECK ((StartTime<EndTime))
```

Clients – Tabela przechowująca informacje o klientach. Dane w niej zawarte to identyfikator klienta (klucz główny), pole do zaznaczenia czy dany klient jest firmą, czy osobą, nazwa firmy (jeśli company=1), nazwisko i imię (w przypadku klienta będącego firmą może być to osoba kontaktowa), dane adresowe oraz numer telefonu.

```
CREATE TABLE Clients (
	ClientID int not null primary key
	identity(1,1), Company bit not null
	DEFAULT(0), CompanyName nvarchar(40) null,
	LastName nvarchar(25) null,
	FirstName nvarchar(25) null,
	Address nvarchar(50) null,
	City nvarchar(40) null,
	PostalCode nvarchar(10) null,
	Phone nvarchar(15) not null CHECK(ISNUMERIC(Phone)=1),
)

ALTER TABLE Clients WITH CHECK ADD CONSTRAINT [CompanyAndCompanyName]
CHECK ((Company=1 AND CompanyName IS NOT NULL) OR (Company=0 AND CompanyName IS NULL))
```

Participants – Tabela przechowuje informacje o uczestnikach konferencji. Zawiera identyfikator uczestnika (klucz główny), identyfikator klienta, który rezerwował miejsce dla uczestnika, nazwisko i imię uczestnika, numer telefonu, pole do oznaczenia czy dana osoba jest studentem i jeśli tak to pole do wprowadzenia numeru jej legitymacji studenckiej.

```
CREATE TABLE Participants (
	ParticipantID int not null primary key identity(1,1),
	ClientID int not null foreign key references Clients(ClientID),
	LastName nvarchar(25) not null,
	FirstName nvarchar(25) not null,
	Phone nvarchar(15) not null CHECK(ISNUMERIC(Phone)=1),
	Student bit not null DEFAULT(0),
	StudentIDNo int null,
)

ALTER TABLE Participants WITH CHECK ADD CONSTRAINT [OnlyStudentWithIDNo]
CHECK ((Student=1 AND StudentIDNo IS NOT NULL) OR (Student=0 AND StudentIDNo IS NULL))
```

ConferenceReservations – Tabela przechowuje informacje o rezerwacjach na poszczególne dni konferencji. Zapisane w niej są identyfikator rezerwacji (klucz główny), klucze obce: identyfikatory klienta i dnia konferencji, data rezerwacji domyślnie ustawiana na dzień, w którym jest ona dokonywana, ilość uczestników, w tym ilość studentów oraz pole do zaznaczenia czy dana rezerwacja została anulowana.

```
CREATE TABLE ConferenceReservations (
	ConfResID int not null primary key identity(1,1),
	DayID int not null foreign key references ConferenceDays(DayID),
	ClientID int not null foreign key references Clients(ClientID),
	ResDate date not null DEFAULT GETDATE(),
	NoOfParticipants int not null CHECK (NoOfParticipants>0),
	NoOfStudents int not null CHECK (NoOfStudents>=0),
	Cancelled bit not null DEFAULT(0),
)

ALTER TABLE ConferenceReservations WITH CHECK ADD
CONSTRAINT [StudentsNotGreaterThanParticipants]
CHECK ((NoOfStudents<=NoOfParticipants))
```

ConfDayRegistrations – Tabela przechowuje infomacje o rejestracjach na konferencje dokonywanych przez pojedynczych uczestników. W niej zapisywane są identyfikatory rejestracji (klucz główny), identyfikatory rezerwacji podawane przez uczestników oraz identyfikatory uczestników rejestrujących się.

```
CREATE TABLE ConfDayRegistrations (
	RegistrationID int not null primary key identity(1,1),
	ReservationID int not null foreign key references
		ConferenceReservations(ConfResID),
	ParticipantID int not null foreign key references Participants(ParticipantID),
)
```

WorkshopReservations – Tabela przechowuje informacje o rezerwacjach na warsztaty. Zawiera identyfikator rezerwacji (klucz główny), identyfikator warsztatu, na który dokonywana jest rezerwacja, identyfikator rejestracji na konferencję (uczestnik warsztatu musi być zarejestrowany na dany dzień), datę rezerwacji oraz pole do zaznaczenia czy dana rezerwacja została anulowana.

```
CREATE TABLE WorkshopReservations (
	WorkResID int not null primary key identity(1,1),
	WorkshopID int not null foreign key references
	Workshops(WorkshopID), DayRegID int not null foreign key references
		ConfDayRegistrations(RegistrationID),
	ResDate date not null DEFAULT GETDATE(),
	Cancelled bit not null DEFAULT(0),
)
```

WorkshopRegistrations – Tabela przechowuje informacje o rejestracjach na warsztaty. Zawiera identyfikator rejestracji (klucz główny) oraz identyfikator rezerwacji, który odnosi się do odpowiedniej rezerwacji warsztatu.

```
CREATE TABLE WorkshopRegistrations (
      RegistrationID int not null primary key identity(1,1),
      ReservationID int not null foreign key references
           WorkshopReservations(WorkResID),
)
```

Payments – Tabela przechowuje informacje o dokonanych opłatach. Posiada identyfikator opłaty (klucz główny), identyfikator klienta dokonującego jej, identyfikator rezerwacji konferencji lub rezerwacji warsztatu (jedno z tych pól jest puste, aby oddzielić opłaty za dni konferencji od tych za warsztaty), ilość pieniędzy, która została zapłacona i datę opłaty.

```
CREATE TABLE Payments (
      PaymentID int not null primary key identity(1,1),
      ClientID int not null foreign key references Clients(ClientID),
      ConfResID int null foreign key references ConferenceReservations(ConfResID),
      WorkResID int null foreign key references WorkshopReservations(WorkResID),
      Paid numeric(2,2) null CHECK (Paid>0),
      PayDate date null DEFAULT GETDATE(),
)

ALTER TABLE PAYMENTS WITH CHECK ADD CONSTRAINT [ConferenceOrWorkshop]
CHECK ((WorkResID IS NULL AND ConfResID IS NOT NULL) OR (WorkResID IS NOT
NULL AND ConfResID IS NULL))
GO
```

5. Indeksy

Indeksy zostały utworzone w tabelach na każdym z kluczy obcych, ponieważ istnieje możliwość że wartości tych pól będą rzadko się powtarzać, a dzięki zastosowaniu nieklastrowanych indeksów wyszukiwanie będzie szybsze.

```
CREATE NONCLUSTERED INDEX ConferenceDaysConferenceID ON ConferenceDays
(
ConferenceID ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB
= OFF, DROP_EXISTING = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)

CREATE NONCLUSTERED INDEX ConferenceReservationsDayID ON ConferenceReservations
(
DayID ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB
= OFF, DROP_EXISTING = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)

CREATE NONCLUSTERED INDEX ConferenceReservationsClientID ON ConferenceReservations
(
ClientID ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB
= OFF, DROP_EXISTING = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)

CREATE NONCLUSTERED INDEX ParticipantsClientID ON Participants
(
ClientID ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB
= OFF, DROP_EXISTING = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
```

```
CREATE NONCLUSTERED INDEX ConfDayRegistrationsReservationID ON ConfDayRegistrations
(
ReservationID ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB =
OFF, DROP_EXISTING = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)

CREATE NONCLUSTERED INDEX ConfDayRegistrationsParticipantID ON ConfDayRegistrations
(
ParticipantID ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB =
OFF, DROP_EXISTING = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)

CREATE NONCLUSTERED INDEX WorkshopReservationsWorkshopID ON WorkshopReservations
(
WorkshopID ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB
= OFF, DROP_EXISTING = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)

CREATE NONCLUSTERED INDEX WorkshopReservationsDayRegID ON WorkshopReservations
(
DayRegID ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB
= OFF, DROP_EXISTING = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)

CREATE NONCLUSTERED INDEX WorkshopsDayID ON Workshops
(
DayID ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB
= OFF, DROP_EXISTING = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)

CREATE NONCLUSTERED INDEX WorkshopsPriceID ON Workshops
(
PriceID ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB
= OFF, DROP_EXISTING = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)

CREATE NONCLUSTERED INDEX PricesDayID ON Prices
(
DayID ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB
= OFF, DROP_EXISTING = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)

CREATE NONCLUSTERED INDEX WorkshopRegistrationReservationID ON WorkshopRegistrations
(
ReservationID ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB
= OFF, DROP_EXISTING = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)

CREATE NONCLUSTERED INDEX PaymentsWorkResID ON Payments
(
WorkResID ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB
= OFF, DROP_EXISTING = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
```

```
CREATE NONCLUSTERED INDEX PaymentsConfResID ON Payments
(
ConfResID ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB
= OFF, DROP_EXISTING = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)

CREATE NONCLUSTERED INDEX PaymentsClientID ON Payments
(
ClientID ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB
= OFF, DROP_EXISTING = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
```

6. Widoki

MostPopularConferences – Widok 100 najpopularniejszych konferencji uporządkowanych malejąco według średniej ilości osób przypadających na jej jeden dzień.

```
CREATE VIEW MostPopularConferences
AS
SELECT TOP 100 c.ConferenceName, c.Address, c.City, c.startDate, c.endDate,
(SUM(cr.NoOfParticipants)/(DATEDIFF(day,c.StartDate,c.EndDate)+1)) AS
AverageParticipantsPerDay
FROM Conferences c INNER JOIN ConferenceDays cd ON c.ConferenceID=cd.ConferenceID
INNER JOIN ConferenceReservations cr ON cd.DayID=cr.DayID AND cr.Cancelled=0
GROUP BY c.ConferenceID, c.ConferenceName, c.Address, c.City, c.startDate, c.endDate
ORDER BY AverageParticipantsPerDay DESC
GO
```

MostPopularWorkshops – Widok 100 najpopularniejszych warsztatów uporządkowanych malejąco według ilości uczestników.

```
CREATE VIEW MostPopularWorkshops
AS
SELECT TOP 100 w.WorkshopName, c.ConferenceName, cd.Date,
COUNT(wr.WorkResID) AS Participants
FROM Conferences c INNER JOIN ConferenceDays cd ON
c.ConferenceID=cd.ConferenceID INNER JOIN Workshops w ON w.DayID=cd.DayID
INNER JOIN WorkshopReservations wr ON wr.WorkshopID=w.WorkshopID AND
wr.Cancelled=0 GROUP BY w.WorkshopID,w.WorkshopName, c.ConferenceName, cd.Date
ORDER BY Participants DESC
GO
```

AvailableDays – Widok prezentujący dni konferencji, które dopiero się odbędą i są jeszcze wolne miejsca.

```
CREATE VIEW AvailableDays
AS
SELECT c.ConferenceName, c.Address, c.City, cd.Date,
cd.MaxPeople,(cd.MaxPeople-SUM(cr.NoOfParticipants)) AS FreePlaces
FROM Conferences c INNER JOIN ConferenceDays cd ON
c.conferenceID=cd.conferenceID AND cd.Date>=GETDATE()
INNER JOIN ConferenceReservations cr ON cd.dayID=cr.DayID AND cr.Cancelled=0
GROUP BY cr.DayID, c.ConferenceName,c.Address, c.City, cd.Date,cd.MaxPeople
GO
```

AvailableWorkshops - Widok prezentujący warsztaty, które dopiero się odbędą i są jeszcze wolne miejsca.

```
CREATE VIEW AvailableWorkshops
AS
SELECT w.WorkshopName, c.ConferenceName, c.Address, c.City, cd.Date,
w.MaxPeople,(w.MaxPeople-COUNT(wr.WorkResID)) AS FreePlaces
FROM Conferences c INNER JOIN ConferenceDays cd ON
c.ConferenceID=cd.ConferenceID AND cd.Date>=GETDATE()
INNER JOIN Workshops w ON w.DayID=cd.DayID
INNER JOIN WorkshopReservations wr ON wr.WorkshopID=w.WorkshopID AND wr.Cancelled=0
GROUP BY w.WorkshopName, c.ConferenceName, c.Address, c.City, cd.Date, w.MaxPeople
GO
```

UnpaidConferenceReservations – Widok wyświetlający dane klientów oraz dni konferencji, na które klienci nie dokonali jeszcze opłat oraz ilość dni pozostałych na opłacenie rezerwacji.

```
CREATE VIEW UnpaidConferenceReservations
AS
SELECT DISTINCT cl.CompanyName, cl.LastName,cl.FirstName, cl.Phone,
c.ConferenceName, cd.Date, (7-DATEDIFF(day,cr.ResDate,GETDATE())) AS DaysLeft
FROM Conferences c INNER JOIN ConferenceDays cd ON
c.ConferenceID=cd.ConferenceID INNER JOIN Prices pr ON cd.DayID=pr.DayID
INNER JOIN ConferenceReservations cr ON cd.DayID=cr.DayID AND
cr.Cancelled=0 INNER JOIN Clients cl ON cr.ClientID=cl.ClientID LEFT OUTER
JOIN Payments p ON p.ClientID=cl.ClientID
WHERE p.PaymentID IS NULL OR p.Paid<((cr.NoOfParticipants-
cr.NoOfStudents)*(pr.Price*(1-pr.Discount))+cr.NoOfStudents*(pr.Price*(1-
pr.Discount-pr.StudentDiscount)))
AND pr.DaysTo=(SELECT TOP 1 pri.DaysTo FROM Prices pri WHERE pri.DayID=pr.DayID AND
DATEDIFF(day,cr.ResDate,cd.Date)>=pri.DaysTo ORDER BY pri.DaysTo) GO
```

UnpaidWorkshopReservations - Widok wyświetlający dane uczestników oraz warsztatów, na które rezerwacje jeszcze nie zostały opłacone i ilość dni pozostałych na opłacenie rezerwacji.

```
CREATE VIEW UnpaidWorkshopReservations
AS
SELECT DISTINCT p.LastName,p.FirstName, p.Phone, w.WorkshopName,
c.ConferenceName, cd.Date, (7-DATEDIFF(day,wr.ResDate,GETDATE())) AS DaysLeft
FROM Conferences c INNER JOIN ConferenceDays cd ON
c.ConferenceID=cd.ConferenceID INNER JOIN Workshops w ON cd.DayID=w.DayID INNER
JOIN Prices pr ON w.PriceID=pr.PriceID
INNER JOIN ConferenceReservations cr ON cd.DayID=cr.DayID AND cr.Cancelled=0
INNER JOIN ConfDayRegistrations cdr ON cr.ConfResID=cdr.ReservationID INNER JOIN
WorkshopReservations wr ON cdr.RegistrationID=wr.DayRegID
INNER JOIN Participants p ON cdr.ParticipantID=p.ParticipantID
LEFT OUTER JOIN Payments pay ON pay.ClientID=p.ParticipantID
WHERE pay.PaymentID IS NULL OR pay.Paid<pr.Price
GO
```

ClientsHistory – Widok pozwalający na przeglądnięcie statystyk klientów: ilość rezerwacji na dni konferencji, łączna ilość uczestników i ilość anulowanych rezerwacji

```
CREATE VIEW ClientsHistory
AS
SELECT cl.CompanyName, cl.LastName, cl.FirstName, COUNT(cr.ConfResID) AS
DaysReservations, SUM(cr.NoOfParticipants) AS SumParticipants,
COUNT(cr.Cancelled) AS CancelledReservations
FROM Clients cl INNER JOIN ConferenceReservations cr ON cl.ClientID=cr.ClientID
GROUP BY cl.CompanyName, cl.LastName, cl.FirstName, cr.Cancelled
GO
```

BestClients – Widok przedstawiający 100 najlepszych klientów, tzn. takich, którzy zarezerwowali łącznie na wszystkie konferencje najwięcej miejsc.

```
CREATE VIEW BestClients
AS
SELECT TOP 100 cl.CompanyName, cl.LastName, cl.FirstName, COUNT(cr.ConfResID) AS
DaysReservations, SUM(cr.NoOfParticipants) AS Participants
FROM Clients cl INNER JOIN ConferenceReservations cr ON
cl.ClientID=cr.ClientID AND cr.Cancelled=0
GROUP BY cl.CompanyName, cl.LastName, cl.FirstName,
cr.Cancelled ORDER BY Participants,DaysReservations DESC GO
```

CancelledConferenceReservations – Widok anulowanych rezerwacji na dni konferencji.

```
CREATE VIEW CancelledConferenceReservations
AS
SELECT cl.ClientID, cr.ConfResID, c.ConferenceName, cd.Date, cr.NoOfParticipants FROM
Conferences c INNER JOIN ConferenceDays cd ON c.ConferenceID=cd.ConferenceID INNER
JOIN ConferenceReservations cr ON cd.DayID=cr.DayID AND cr.Cancelled=1 INNER JOIN
Clients cl ON cr.ClientID=cl.ClientID
GO
```

CancelledWorshopReservations – Widok anulowanych rezerwacji na warsztaty.

```
CREATE VIEW CancelledWorkshopReservations
AS
SELECT p.ParticipantID, wr.WorkResID, c.ConferenceName, w.WorkshopName, cd.Date FROM
Conferences c INNER JOIN ConferenceDays cd ON c.ConferenceID=cd.ConferenceID INNER
JOIN ConferenceReservations cr ON cd.DayID=cr.DayID
INNER JOIN ConfDayRegistrations cdr ON cr.ConfResID=cdr.ReservationID INNER
JOIN WorkshopReservations wr ON cdr.RegistrationID=wr.DayRegID AND
wr.Cancelled=1 INNER JOIN Workshops w ON wr.WorkshopID=wr.WorkshopID INNER
JOIN Participants p ON cdr.ParticipantID=p.ParticipantID
GO
```

ConferenceDaysReservedButNotRegistered – Widok rezerwacji na dni konferencji, które jeszcze nie zostały jeszcze w całości wypełnione zarejestrowanymi uczestnikami.

```
CREATE VIEW ConferenceDaysReservedButNotRegistered
AS
SELECT c.ConferenceName, cd.Date, cl.CompanyName, cl.LastName, cl.FirstName,
DATEDIFF(day,GETDATE(),c.StartDate) AS DaysLeftForRegistration,
cr.NoOfParticipants, COUNT (cdr.RegistrationID) AS RegisteredNow
FROM Conferences c INNER JOIN ConferenceDays cd ON
c.ConferenceID=cd.ConferenceID INNER JOIN ConferenceReservations cr
ON cd.DayID=cr.DayID INNER JOIN Clients cl ON cr.clientID=cl.ClientID
LEFT OUTER JOIN ConfDayRegistrations cdr ON cr.ConfResID=cdr.ReservationID
WHERE (SELECT COUNT(cdr2.RegistrationID) FROM ConfDayRegistrations cdr2 WHERE
cdr.RegistrationID=cdr2.RegistrationID)<cr.NoOfParticipants
GROUP BY c.ConferenceName, cd.Date, cl.CompanyName,
cl.LastName, cl.FirstName, c.StartDate,cr.NoOfParticipants
GO
```

UpcomingConfDaysPrices – Widok przedstawiający nadchodzące dni konferencji i ich ceny za miejsce wraz ze zniżkami.

```
CREATE VIEW UpcomingConfDaysPrices
AS
SELECT c.ConferenceName, cd.Date, p.Price, p.DaysTo, p.Discount,
p.StudentDiscount FROM Conferences c INNER JOIN ConferenceDays cd ON
c.ConferenceID=cd.ConferenceID INNER JOIN Prices p ON p.DayID=cd.DayID
WHERE p.DaysTo<DATEDIFF(day,GETDATE(),cd.Date)
GO
```

OverpaidConferences – Widok przedstawiający klientów i ich rezerwacje dni konferencji, za które zapłacili za dużo.

```
CREATE VIEW OverpaidConferences
AS
SELECT
c.ConferenceName,cd.Date,cl.ClientID,cl.CompanyName,cl.LastName,cl.FirstName,cl.Phone,
(pay.Paid-(cr.NoOfParticipants-cr.NoOfStudents)*(p.Price*(1-
p.Discount))+cr.NoOfStudents*(p.Price*(1-p.Discount-p.StudentDiscount))) AS Difference
FROM Conferences c INNER JOIN ConferenceDays cd ON c.ConferenceID=cd.ConferenceID
INNER JOIN ConferenceReservations cr ON cd.DayID=cr.DayID
INNER JOIN Clients cl ON cr.ClientID=cl.ClientID
INNER JOIN Prices p ON p.DayID=cd.DayID AND p.DaysTo=(SELECT TOP 1 pri.DaysTo FROM
Prices pri WHERE pri.DayID=p.DayID AND
DATEDIFF(day,cr.ResDate,cd.Date)>=pri.DaysTo ORDER BY pri.DaysTo)
INNER JOIN Payments pay ON cr.ConfResID=pay.ConfResID AND cl.ClientID=pay.ClientID
WHERE (pay.Paid-(cr.NoOfParticipants-cr.NoOfStudents)*(p.Price*(1-
p.Discount))+cr.NoOfStudents*(p.Price*(1-p.Discount-p.StudentDiscount)))>0
GO
```

OverpaidWorkshops - Widok przedstawiający uczestników i ich rezerwacje na warsztaty, za które zapłacili za dużo.

```
CREATE VIEW OverpaidWorkshops
AS
SELECT c.ConferenceName,
w.WorkshopName,cd.Date,p.ParticipantID,p.LastName,p.FirstName,p.Phone,
(pay.Paid-pr.Price) AS Difference
FROM Conferences c INNER JOIN ConferenceDays cd ON c.ConferenceID=cd.ConferenceID
INNER JOIN ConferenceReservations cr ON cd.DayID=cr.DayID AND cr.Cancelled=0 INNER
JOIN ConfDayRegistrations cdr ON cr.ConfResID=cdr.ReservationID
INNER JOIN WorkshopReservations wr ON cdr.RegistrationID=wr.DayRegID AND
wr.Cancelled=0 INNER JOIN Workshops w ON wr.WorkshopID=w.WorkshopID
INNER JOIN Participants p ON cdr.ParticipantID=p.ParticipantID
INNER JOIN Prices pr ON w.PriceID=pr.PriceID
INNER JOIN Payments pay ON pay.WorkResID=wr.WorkResID
WHERE (pay.Paid-pr.Price)>0
GO
```

CurrentReservationsRegisteredParticipants – Widok przedstawiający rezerwacje na nadchodzące dni konferencji oraz liczbę zarejestrowanych uczestników w ramach tej rezerwacji.

```
CREATE VIEW CurrentReservationsRegisteredParticipants
AS
SELECT cl.ClientID, cl.CompanyName, cl.Firstname, cl.Lastname, cr.ConfResID,
c.ConferenceName, cd.Date, cr.NoOfParticipants, COUNT(cdr.RegistrationID) AS
Registered FROM Conferences c INNER JOIN ConferenceDays cd ON
c.ConferenceID=cd.ConferenceID AND cd.Date>=GETDATE()
INNER JOIN ConferenceReservations cr ON cd.DayID=cr.DayID AND Cancelled=0
INNER JOIN ConfDayRegistrations cdr ON cr.ConfResID=cdr.ReservationID INNER
JOIN Clients cl ON cr.ClientID=cl.ClientID
GROUP BY cl.ClientID, cl.CompanyName, cl.Firstname, cl.Lastname,
cr.ConfResID, c.ConferenceName, cd.Date, cr.NoOfParticipants
GO
```

7. Triggery

ClearTablesAfterCancellingConfRes – Usuwa dane z tabel powiązanych z rezerwacją konferencji, która została anulowana (rejestracje na konferencję, rezerwacje na warsztaty, rejestracje na warsztaty)

```
CREATE TRIGGER ClearTablesAfterCancellingConfRes
ON ConferenceReservations
AFTER UPDATE
AS
BEGIN
	DELETE FROM WorkshopRegistrations
		WHERE RegistrationID IN (SELECT wreg.RegistrationID FROM
		WorkshopRegistrations wreg INNER JOIN WorkshopReservations wr ON
		wreg.ReservationID=wr.WorkResID INNER JOIN ConfDayRegistrations cdr ON
		wr.DayRegID=cdr.RegistrationID INNER JOIN ConferenceReservations cr ON
		cdr.RegistrationID=cr.ConfResID WHERE cr.Cancelled=1
					)

	DELETE FROM WorkshopReservations
		WHERE WorkResID IN (SELECT wr.WorkResID FROM WorkshopReservations wr
		INNER JOIN ConfDayRegistrations cdr ON wr.DayRegID=cdr.RegistrationID
		INNER JOIN ConferenceReservations cr ON
		cdr.RegistrationID=cr.ConfResID WHERE cr.Cancelled=1
				)
	DELETE FROM ConfDayRegistrations
		WHERE RegistrationID IN (SELECT cdr.RegistrationID FROM
		ConfDayRegistrations cdr INNER JOIN ConferenceReservations cr
		ON cdr.RegistrationID=cr.ConfResID WHERE cr.Cancelled=1
					)
END
GO
```

DeleteRegistrationAfterCancellingWorkRes – Usuwa rejestrację powiązaną z rezerwacją na warsztat, która została anulowana

```
CREATE TRIGGER DeleteRegistrationAfterCancellingWorkRes
ON WorkshopReservations
AFTER UPDATE
AS
BEGIN
	DELETE FROM WorkshopRegistrations
		WHERE RegistrationID IN (SELECT wreg.RegistrationID FROM
		WorkshopRegistrations wreg INNER JOIN WorkshopReservations wr
		ON wreg.ReservationID=wr.WorkResID WHERE wr.Cancelled=1
					)
END
GO
```

CheckWorkshopReservations – Sprawdza spójność danych po wprowadzeniu rezerwacji na warsztat (czy uczestnik nie zapisał się na dwa warsztaty odbywające się w tym samym czasie oraz czy nie zapisał się drugi raz na ten sam warsztat)

```
CREATE TRIGGER CheckWorkshopReservations
ON WorkshopReservations
AFTER INSERT
AS
BEGIN
	IF EXISTS
		(SELECT * FROM Inserted iwr INNER JOIN Workshops iw
		ON iwr.WorkshopID=iw.WorkshopID
		INNER JOIN ConfDayRegistrations icdr ON iwr.DayRegID=icdr.RegistrationID
		INNER JOIN Participants ip ON icdr.ParticipantID=ip.ParticipantID
		WHERE iwr.WorkResID IN(SELECT wr.WorkResID FROM WorkshopReservations wr
		INNER JOIN Workshops w ON wr.WorkshopID=w.WorkshopID
		INNER JOIN ConfDayRegistrations cdr ON iwr.DayRegID=cdr.RegistrationID INNER
		JOIN Participants p ON cdr.ParticipantID=p.ParticipantID
		WHERE w.WorkshopID<>iw.WorkshopID AND ip.ParticipantID=p.ParticipantID AND
		((iw.StartTime BETWEEN w.StartTime AND w.EndTime) OR (iw.StartTime BETWEEN
		w.StartTime AND w.EndTime))
				)
		)
	BEGIN
		RAISERROR ('Can''t make a reservation for two workshops in the
		same time',14,1)
		ROLLBACK TRANSACTION
	END
	IF EXISTS
		(SELECT COUNT(wr.WorkResID) FROM WorkshopReservations wr INNER JOIN
		ConfDayRegistrations cdr ON wr.DayRegID=cdr.RegistrationID
		INNER JOIN Participants p ON cdr.ParticipantID=p.ParticipantID
		GROUP BY wr.WorkshopID, p.ParticipantID
		HAVING COUNT(wr.WorkResID)>1
		)
	BEGIN
		RAISERROR ('Can''t make two reservations for the same workshop',14,1)
		ROLLBACK TRANSACTION
	END
END
GO
```

CheckConfDayRegistrations – Sprawdza spójność danych po rejestracji na konferencję (czy uczestnik nie rejestrował się dwa razy w ramach tej samej rezerwacji lub czy nie zarejestrował się na dwie konferencje odbywające się w ten sam dzień)

```
CREATE TRIGGER CheckConfDayRegistrations
ON ConfDayRegistrations
AFTER INSERT
AS
BEGIN
	IF EXISTS
		(SELECT * FROM Inserted icdr INNER JOIN ConferenceReservations icr
		ON cdr.ReservationID=icr.ConfResID
		INNER JOIN ConferenceDays icd ON icr.DayID=icd.DayID
		INNER JOIN Participants ip ON
		icdr.ParticipantID=ip.ParticipantID WHERE icdr.RegistrationID IN
		(SELECT cdr.RegistrationID FROM ConfDayRegistrations cdr INNER JOIN
		ConferenceReservations cr ON cdr.ReservationID=cr.ConfResID INNER
		JOIN ConferenceDays cd ON cr.DayID=cd.DayID
		INNER JOIN Participants p ON cdr.ParticipantID=p.ParticipantID
		WHERE cr.ConfResID<>icr.ConfResID AND ip.ParticipantID=p.ParticipantID AND
		icd.Date=cd.Date)
		)
	BEGIN
		RAISERROR ('Can''t register to two conferences on the same day',14,1)
		ROLLBACK TRANSACTION
	END
	IF EXISTS
		(SELECT COUNT(*) FROM Participants p INNER JOIN ConfDayRegistrations cdr
		ON cdr.ParticipantID=p.ParticipantID
		GROUP BY cdr.ReservationID,p.ParticipantID
		HAVING COUNT (p.ParticipantID)>1
		)
	BEGIN
		RAISERROR ('Can''t register twice on the same Reservation ID',14,1)
		ROLLBACK TRANSACTION
	END
END
GO
```

CheckConfDayMaxPeople – Sprawdza czy po dokonaniu rezerwacji nie zostanie przekroczony limit miejsc na dany dzień konferencji

```
CREATE TRIGGER CheckConfDayMaxPeople
ON ConferenceReservations
AFTER INSERT
AS
BEGIN
	IF EXISTS
		(SELECT SUM(cr.NoOfParticipants) FROM ConferenceReservations cr INNER
		JOIN ConferenceDays cd
			ON cr.DayID=cd.DayID
			WHERE cr.Cancelled=0
			GROUP BY cd.DayID
			HAVING SUM(cr.NoOfParticipants)>(SELECT cd2.MaxPeople
			FROM ConferenceDays cd2 WHERE cd2.DayID=cd.DayID)
		)
	BEGIN
		RAISERROR ('Not enough places on this day',14,1)
		ROLLBACK TRANSACTION
	END
END
GO
```

CheckWorkshopMaxPeople – Sprawdza czy po dokonaniu rezerwacji nie zostanie przekroczony limit miejsc na dany warsztat

```
CREATE TRIGGER CheckWorkshopMaxPeople
ON WorkshopReservations
AFTER INSERT
AS
BEGIN
	IF EXISTS
		(SELECT COUNT(WorkResID) FROM WorkshopReservations wr INNER
		JOIN Workshops w ON wr.WorkshopID=w.WorkshopID WHERE
		wr.Cancelled=0 GROUP BY w.WorkshopID
		HAVING COUNT(WorkResID)>(SELECT w2.MaxPeople FROM Workshops w2 WHERE
		w.WorkshopID=w2.WorkshopID)
		)
	BEGIN
		RAISERROR ('Not enough places on this workshop',14,1)
		ROLLBACK TRANSACTION
	END
END
GO
```

CheckWorkshopPriceNotDayPrice – Sprawdza czy klucz obcy w tabeli warsztatów nie odwołuje się do ceny za dzień konferencji (ceny za warsztat nie mają DayID)

```
CREATE TRIGGER
CheckWorkshopPriceNotDayPrice ON Workshops
AFTER INSERT,UPDATE
AS
BEGIN
	IF EXISTS (SELECT w.WorkshopID FROM Workshops w INNER JOIN Prices ON
			w.PriceID=p.PriceID AND p.DayID IS NOT NULL)
	BEGIN
		RAISERROR ('Workshop price must have ID different than day price',14,1)
		ROLLBACK TRANSACTION
	END
END
GO
```

CheckNoOfRegisteredParticipants – Sprawdza czy po zarejstrowaniu kolejnego uczestnika w ramach jednej rezerwacji nie zostanie przekroczona ilość zarezerwowanych miejsc

```
CREATE TRIGGER CheckNoOfRegisteredParticipants ON
ConfDayRegistrations
AFTER INSERT
AS
BEGIN
	IF EXISTS(SELECT * FROM Inserted icdr INNER JOIN ConferenceReservations icr
		ON icdr.RegistrationID=icr.ConfResID WHERE icr.ConfResID IN (SELECT
		cr.ConfResID FROM ConfDayRegistrations cdr INNER JOIN
		ConferenceReservations cr ON cdr.ReservationID=cr.ConfResID
		WHERE (SELECT COUNT (cdr2.RegistrationID) FROM ConfDayRegistrations cdr2
		WHERE cdr2.ReservationID=cr.ConfResID)>cr.NoOfParticipants)
			)
	BEGIN
		RAISERROR ('No more places available for this reservation',14,1)
		ROLLBACK TRANSACTION
	END
END
```

8. Procedury

AddConferenceDay – Dodawanie dnia konferencji

```
CREATE PROCEDURE AddConferenceDay
	@ConferenceID int,
	@Date date,
	@MaxPeople int
AS
BEGIN
	INSERT INTO ConferenceDays(
		ConferenceID, Date, MaxPeople
	)
	VALUES(
		@ConferenceID, @Date, @MaxPeople
	)
END
GO
```

AddConference – Dodawanie konferencji

```
CREATE PROCEDURE AddConference
	@ConferenceName nvarchar(60),
	@Address nvarchar (50),
	@City nvarchar(40),
	@PostalCode nvarchar(10),
	@StartDate date,
	@EndDate date,
	@MaxPeople int
AS
BEGIN
	SET NOCOUNT ON;
	IF (@StartDate<GETDATE())
	BEGIN
		RAISERROR('Conference can''t start in the past',14,1)
		RETURN
	END
	INSERT INTO Conferences(
		ConferenceName, Address, City, PostalCode, StartDate, EndDate
	)
	VALUES(
		@ConferenceName, @Address, @City, @PostalCode, @StartDate, @EndDate
	)
	DECLARE @ConferenceID int
	SET @ConferenceID=@@IDENTITY
	DECLARE @i int
	SET @i=0
	DECLARE @d date
	WHILE @i<=DATEDIFF(day,@StartDate,@EndDate)
	BEGIN
		SET @d=DATEADD(day,@i,@StartDate)
		EXEC AddConferenceDay
			@ConferenceID,
			@d,
			@MaxPeople
		SET @i=@i+1
	END
END
GO
```

AddPrice – Dodawanie ceny (progu)

```
CREATE PROCEDURE AddPrice
	@DayID int,
	@Price numeric(2,2),
	@DaysTo int,
	@Discount numeric(2,2),
	@StudentDiscount numeric (2,2)
AS
BEGIN

	SET NOCOUNT ON;
	INSERT INTO Prices(
		Price,DaysTo,Discount,StudentDiscount, DayID
	)
	VALUES(
		@Price,@DaysTo,@Discount,@StudentDiscount,@DayID
	)
END
GO
```

AddWorkshop – Dodawanie warsztatu

```
CREATE PROCEDURE AddWorkshop
	@WorkshopName nvarchar(60),
	@DayID int ,
	@StartTime time,
	@EndTime time,
	@MaxPeople int,
	@PriceID int
AS
BEGIN
	SET NOCOUNT ON;
	INSERT INTO Workshops(
		WorkshopName,DayID,StartTime,EndTime,MaxPeople,PriceID
	)
	VALUES (
		@WorkshopName, @DayID, @StartTime, @EndTime, @MaxPeople, @PriceID
	)
END
GO
```

AddClient – Dodawanie klienta

```
CREATE PROCEDURE AddClient
	@Company bit,
	@CompanyName nvarchar(40),
	@LastName nvarchar(25),
	@FirstName nvarchar(25),
	@Address nvarchar(50),
	@City nvarchar(40),
	@PostalCode nvarchar(10),
	@Phone nvarchar(15)
AS
BEGIN
	SET NOCOUNT ON;
	INSERT INTO Clients(
		Company,CompanyName,LastName,FirstName,Address,City,PostalCode,Phone
	)
	VALUES(

	@Company,@CompanyName,@LastName,@FirstName,@Address,@City,@PostalCode,@Phone
	)
END
GO
```

AddParticipant – Dodawanie uczestnika

```
CREATE PROCEDURE AddParticipant
	@ClientID int,
	@LastName nvarchar(25),
	@Firstname nvarchar(25),
	@Phone nvarchar(15),
	@Student bit,
	@StudentIDNo integer
AS
BEGIN
	SET NOCOUNT ON;
	INSERT INTO Participants(
		ClientID, LastName, FirstName, Phone, Student, StudentIDNo
	)
	VALUES(
		@ClientID, @LastName, @FirstName, @Phone, @Student, @StudentIDNo
	)
END
GO
```

MakeConfDayReservation – Dokonanie rezerwacji na dzień konferencji

```
CREATE PROCEDURE MakeConfDayReservation
	@DayID int,
	@ClientID int,
	@NoOfParticipants int,
	@NoOfStudents int
AS
BEGIN
	IF GETDATE()>(SELECT Date FROM ConferenceDays WHERE DayID=@DayID)
	BEGIN
		RAISERROR('Can''t make reservation for past conference day',14,1)
		RETURN
	END
	SET NOCOUNT ON;
	INSERT INTO ConferenceReservations(
		DayID, ClientID, NoOfParticipants,NoOfStudents
	)
	VALUES (
		@DayID, @ClientID, @NoOfParticipants,@NoOfStudents
	)
END
GO
```

MakeConferenceReservation – Dokonanie rezerwacji na całą konferencję (każdy z jej dni)

```
CREATE PROCEDURE MakeConferenceReservation
	@ConferenceID int,
	@ClientID int,
	@NoOfParticipants int,
	@NoOfStudents int
AS
BEGIN
	IF GETDATE()>(SELECT StartDate FROM Conferences WHERE ConferenceID=@ConferenceID)
	BEGIN
		RAISERROR('Can''t make reservation for past conference',14,1)
		RETURN
	END
	SET NOCOUNT ON;
	DECLARE @i int
	SET @i=0
	WHILE @i<=DATEDIFF(day,(SELECT StartDate FROM Conferences WHERE
	ConferenceID=@ConferenceID),(SELECT EndDate FROM Conferences
	WHERE ConferenceID=@ConferenceID))
	BEGIN
		DECLARE @DayID int
		SET @DayID = (SELECT DayID FROM ConferenceDays cd INNER JOIN Conferences
		c ON cd.ConferenceID=c.ConferenceID WHERE c.ConferenceID=@ConferenceID
		AND cd.Date=DATEADD(day,@i,c.StartDate))
		EXECUTE MakeConfDayReservation
			@DayID,
			@ClientID,
			@NoOfParticipants,
			@NoOfStudents
		SET @i=@i+1
	END
END
GO
```

MakeWorkshopReservation – Dokonanie rezerwacji na warsztat

```
CREATE PROCEDURE MakeWorkshopReservation
	@WorkshopID int,
	@DayRegID int
AS
BEGIN
	IF GETDATE()>(SELECT cd.Date FROM Workshops w INNER JOIN ConferenceDays cd
	ON w.DayID=cd.DayID WHERE WorkshopID=@WorkshopID)
	BEGIN
		RAISERROR('Can''t make reservation for past workshop',14,1)
		RETURN
	END
	SET NOCOUNT ON;
	INSERT INTO WorkshopReservations (
		WorkshopID, DayRegID
	)
	VALUES (
		@WorkshopID, @DayRegID
	)
END
GO
```

RegisterToConferenceDay – Dokonanie rejestracji na dzień konferencji

```
CREATE PROCEDURE RegisterToConferenceDay
      @ReservationID int,
      @ParticipantID int,
      @ClientID int,
      @LastName nvarchar(25),
      @Firstname nvarchar(25),
      @Student bit,
      @StudentIDNo integer
AS
BEGIN
      IF ((SELECT cr.Cancelled FROM ConfDayRegistrations cdr INNER
      JOIN ConferenceReservations cr ON cdr.ReservationID=cr.ConfResID
      WHERE cr.ConfResID=@ReservationID)=1)
      BEGIN
            RAISERROR ('Can''t register. Reservation was cancelled.',14,1)
            RETURN
      END
      SET NOCOUNT ON;
      IF (@ParticipantID IS NULL)
      BEGIN
            EXECUTE AddParticipant
                  @ClientID,
                  @LastName,
                  @Firstname,
                  @Student,
                  @StudentIDNo
                  SET @ParticipantID=@@IDENTITY
      END
      INSERT INTO ConfDayRegistrations (
            ReservationID, ParticipantID
      )
      VALUES (
            @ReservationID, @ParticipantID
      )
END
GO
```

RegisterToWorkshop – Dokonanie rejestracji na warsztat

```
CREATE PROCEDURE RegisterToWorkshop
      @ReservationID int
AS
BEGIN
      IF (SELECT wr.Cancelled FROM WorkshopReservations wr INNER JOIN Workshops
      w ON wr.WorkshopID=w.WorkshopID WHERE wr.WorkResID=@ReservationID)=1
      BEGIN
            RAISERROR ('Can''t register. Reservation was cancelled',14,1)
            RETURN
      END
      SET NOCOUNT ON;
      INSERT INTO WorkshopRegistrations (
            ReservationID
      )
      VALUES (
            @ReservationID
      )
END
GO
```

MakeConfResPayment – Opłacenie rezerwacji na konferencję

```
CREATE PROCEDURE MakeConfResPayment
      @ClientID int,
      @ConfResID int,
      @Paid numeric(2,2)
AS
BEGIN
      SET NOCOUNT ON;
      INSERT INTO Payments(
            ClientID, ConfResID, Paid
      )
      VALUES (
            @ClientID, @ConfResID, @Paid
      )
END
GO
```

MakeWorkResPayment – Opłacenie rezerwacji na warsztat

```
CREATE PROCEDURE MakeWorkResPayment
      @ClientID int,
      @WorkResID int,
      @Paid numeric(2,2)
AS
BEGIN
      SET NOCOUNT ON;
      INSERT INTO Payments(
            ClientID, WorkResID, Paid
      )
      VALUES (
            @ClientID, @WorkResID, @Paid
      )
END
GO
```

PayForWholeReservation – Opłata za całą rezerwację na dzień konferencji

```
CREATE PROCEDURE PayForWholeReservation
      @ClientID int,
      @ConfResID int
AS
BEGIN
      SET NOCOUNT ON;
      DECLARE @Paid int
      SET @Paid=(SELECT (p.Price*(cr.NoOfParticipants-cr.NoOfStudents)*(1-
            Discount)+p.Price*cr.NoOfStudents*(1-StudentDiscount-Discount)) FROM
            Clients cl INNER JOIN ConferenceReservations cr ON
            cl.ClientID=cr.ClientID INNER JOIN ConferenceDays cd ON cr.DayID=cd.DayID
            INNER JOIN Prices p ON p.DayID=cd.DayID
            WHERE ConfResID=@ConfResID AND p.DaysTo=(SELECT TOP 1 DaysTo FROM
            Prices WHERE DATEDIFF(day,cr.ResDate,cd.Date)>=DaysTo ORDER BY DaysTo)
            )
      INSERT INTO Payments(
            ClientID, ConfResID, Paid
      )
      VALUES (
            @ClientID, @ConfResID, @Paid
      )
END
GO
```

GenerateIdentifiers – Wygenerowanie identyfikatorów uczestników danej konferencji

```
CREATE PROCEDURE GenerateIdentifiers

      @ConferenceID int
AS
BEGIN
      SELECT DISTINCT (CONVERT(nvarchar(8),p.ParticipantID)+':
      '+p.FirstName+' '+p.LastName+', '+cl.CompanyName) AS Identifier
      FROM ConferenceDays cd INNER JOIN ConferenceReservations cr ON
      cd.DayID=cr.DayID AND Cancelled=0
      INNER JOIN ConfDayRegistrations cdr ON cr.ConfResID=cdr.ReservationID
      INNER JOIN Clients cl ON cr.ClientID=cl.ClientID INNER JOIN Participants
      p ON cl.ClientID=p.ClientID
      INNER JOIN ConfDayRegistrations cdr2 ON p.ParticipantID=cdr2.ParticipantID
      WHERE cd.ConferenceID=@ConferenceID AND cl.Company=1
      UNION
      SELECT DISTINCT(CONVERT(nvarchar(8),p.ParticipantID)+': '+p.FirstName+'
      ' +p.LastName) AS Identifier
      FROM ConferenceDays cd INNER JOIN ConferenceReservations cr ON
      cd.DayID=cr.DayID AND Cancelled=0
      INNER JOIN ConfDayRegistrations cdr ON cr.ConfResID=cdr.ReservationID
      INNER JOIN Clients cl ON cr.ClientID=cl.ClientID INNER JOIN Participants
      p ON cl.ClientID=p.ClientID
      INNER JOIN ConfDayRegistrations cdr2 ON p.ParticipantID=cdr2.ParticipantID
      WHERE cd.ConferenceID=@ConferenceID AND cl.Company=0
END
GO
```

MyConferences – Sprawdzenie konferencji, na które zarejestrowany jest dany uczestnik

```
 CREATE PROCEDURE MyConferences
      @ParticipantID int
AS
BEGIN
      SELECT c.ConferenceName,c.Address,c.City,c.PostalCode,cd.Date
      FROM Conferences c INNER JOIN ConferenceDays cd ON c.ConferenceID=cd.ConferenceID
      INNER JOIN ConferenceReservations cr ON cd.DayID=cr.DayID AND cr.Cancelled=0 INNER
      JOIN ConfDayRegistrations cdr ON cr.ConfResID=cdr.ReservationID
      INNER JOIN Participants p ON cdr.ParticipantID=p.ParticipantID
      INNER JOIN Prices pr ON cd.DayID=pr.DayID
      AND pr.DaysTo=(SELECT TOP 1 pr2.DaysTo FROM Prices pr2 WHERE pr2.DayID=pr.DayID AND
      DATEDIFF(day,cr.ResDate,cd.Date)>=pr2.DaysTo ORDER BY pr2.DaysTo)
      INNER JOIN Payments pay ON pay.ConfResID=cr.ConfResID
      WHERE p.ParticipantID=@ParticipantID
END
GO
```

MyWorkshops – Sprawdzenie warsztatów, na które zajerestrowany jest dany uczestnik

```
CREATE PROCEDURE MyWorkshops
      @ParticipantID int
AS
BEGIN
      SELECT w.WorkshopName, c.ConferenceName, c.Address,
      c.City,c.PostalCode,cd.Date,w.StartTime,w.EndTime
      FROM Conferences c INNER JOIN ConferenceDays cd ON c.ConferenceID=cd.ConferenceID
      INNER JOIN ConferenceReservations cr ON cd.DayID=cr.DayID AND cr.Cancelled=0 INNER
      JOIN ConfDayRegistrations cdr ON cr.ConfResID=cdr.ReservationID
      INNER JOIN WorkshopReservations wr ON cdr.RegistrationID=wr.DayRegID
      AND wr.Cancelled=0
      INNER JOIN Workshops w ON wr.WorkshopID=w.WorkshopID
      INNER JOIN WorkshopRegistrations wreg ON wr.WorkResID=wreg.ReservationID
      INNER JOIN Participants p ON cdr.ParticipantID=p.ParticipantID INNER JOIN
      Prices pr ON w.PriceID=pr.PriceID
      INNER JOIN Payments pay ON pay.WorkResID=wr.WorkResID
      WHERE p.ParticipantID=@ParticipantID
END
GO
```

ConferenceDayList – Wyświetlenie listy uczestników dnia konferencji

```
CREATE PROCEDURE ConferenceDayList
      @DayID int
AS
BEGIN
      SELECT p.ParticipantID,p.LastName,p.FirstName
      FROM ConferenceDays cd INNER JOIN ConferenceReservations cr ON
      cd.DayID=cr.DayID AND cr.Cancelled=0
      INNER JOIN ConfDayRegistrations cdr ON
      cr.ConfResID=cdr.ReservationID INNER JOIN Participants p ON
      cdr.ParticipantID=p.ParticipantID WHERE @DayID=cd.DayID
END
GO
```

WorkshopList – Wyświetlenie listy uczestników warsztatu

```
CREATE PROCEDURE WorkshopList
      @WorkshopID int
AS
BEGIN
      SELECT p.ParticipantID,p.LastName,p.FirstName
      FROM WorkshopReservations wr
      INNER JOIN WorkshopRegistrations wreg ON wr.WorkResID=wreg.ReservationID
      AND wr.Cancelled=0
      INNER JOIN ConfDayRegistrations cdr ON
      wr.DayRegID=cdr.RegistrationID INNER JOIN Participants p ON
      cdr.ParticipantID=p.ParticipantID WHERE @WorkshopID=wr.WorkshopID
END
GO
```

ClientConfPayments – Wyświetlenie płatności klienta za zarezerwowane dni konferencji

```
CREATE PROCEDURE ClientConfPayments
      @ClientID int
AS
BEGIN
      SELECT c.ConferenceName, cd.Date, cr.NoOfParticipants, cr.NoOfStudents,
      ((cr.NoOfParticipants-cr.NoOfStudents)*(p.Price*(1-p.Discount))
      +cr.NoOfStudents*(p.Price*(1-p.Discount-p.StudentDiscount))) AS ToPay,pay.Paid
      FROM Clients cl INNER JOIN ConferenceReservations cr ON cl.ClientID=cr.ClientID
      INNER JOIN ConferenceDays cd ON cr.DayID=cd.DayID
      INNER JOIN Conferences c ON cd.ConferenceID=c.ConferenceID
      INNER JOIN Prices p ON p.DayID=cd.DayID
      AND p.DaysTo=(SELECT TOP 1 pr.DaysTo FROM Prices pr WHERE pr.DayID=p.DayID
      AND DATEDIFF(day,cr.ResDate,cd.Date)>=pr.DaysTo ORDER BY pr.DaysTo)
      LEFT OUTER JOIN Payments pay ON
      cr.ConfResID=pay.ConfResID WHERE cl.ClientID=@ClientID
END
GO
```

ChangeConferenceDayPlaces – Zmiana ilości miejsc na dany dzień konferencji

```
CREATE PROCEDURE ChangeConferenceDayPlaces
      @ConferenceDayID int,
      @NewMaxPeople int
AS
BEGIN
      SET NOCOUNT ON;
      IF NOT EXISTS (SELECT * FROM ConferenceDays WHERE @ConferenceDayID=DayID)
      BEGIN
            RAISERROR ('No conference day with given ID',14,1)
            RETURN
      END
      IF (@NewMaxPeople<(SELECT SUM(NoOfParticipants) FROM ConferenceReservations
            cr INNER JOIN ConferenceDays cd ON cr.DayID=cd.DayID WHERE
            cd.DayID=@ConferenceDayID))
      BEGIN
            RAISERROR ('Number of reserved places is bigger than new
            participants limit',14,1)
            RETURN
      END
      UPDATE ConferenceDays
            SET MaxPeople=@NewMaxPeople
            WHERE DayID=@ConferenceDayID
END
GO
```

ChangeWorkshopPlaces – Zmiana ilości miejsc na dany warsztat

```
CREATE PROCEDURE ChangeWorkshopPlaces
	@WorkshopID int,
	@NewMaxPeople int
AS
BEGIN
	SET NOCOUNT ON;
	IF NOT EXISTS (SELECT * FROM Workshops WHERE @WorkshopID=WorkshopID)
	BEGIN
		RAISERROR ('No workshop with given ID',14,1)
		RETURN
	END
	IF (@NewMaxPeople<(SELECT COUNT(WorkResID) FROM WorkshopReservations
	wr INNER JOIN Workshops w ON w.WorkshopID=wr.WorkshopID WHERE
	w.WorkshopID=@WorkshopID))
	BEGIN
		RAISERROR ('Number of reserved places is bigger than new
		participants limit',14,1)
		RETURN
	END
	UPDATE Workshops
		SET MaxPeople=@NewMaxPeople WHERE WorkshopID=@WorkshopID
END
GO
```

ChangeConferenceDayReservation – Zmiana ilości zarezerwowanych miejsc na dany dzień konferencji

```
CREATE PROCEDURE ChangeConferenceDayReservation
	@ConfResID int,
	@NewNoOfParticipants int,
	@NewNoOfStudents int
AS
BEGIN
	SET NOCOUNT ON;
	IF NOT EXISTS (SELECT * FROM ConferenceReservations WHERE ConfResID=@ConfResID)
	BEGIN
		RAISERROR ('No reservation with given ID',14,1)
		RETURN
	END
	IF (@NewNoOfParticipants<(SELECT COUNT(cdr.RegistrationID) FROM
	ConfDayRegistrations cdr INNER JOIN ConferenceReservations cr ON
	cdr.ReservationID=cr.ConfResID WHERE cr.ConfResID=@ConfResID))
	BEGIN
		RAISERROR ('Number of registered participants is bigger than new number
		of reserved places',14,1)
		RETURN
	END
	IF (@NewNoOfStudents<(SELECT COUNT(cdr.RegistrationID) FROM ConfDayRegistrations
	cdr INNER JOIN ConferenceReservations cr ON cdr.ReservationID=cr.ConfResID
	INNER JOIN Participants p ON cdr.ParticipantID=p.ParticipantID WHERE p.Student=1
	AND cr.ConfResID=@ConfResID))
	BEGIN
		RAISERROR ('Number of registered students is bigger than new number
		of places for students',14,1)
		RETURN
	END
	UPDATE ConferenceReservations
		SET NoOfParticipants=@NewNoOfParticipants
		WHERE ConfResID=@ConfResID
	UPDATE ConferenceReservations
		SET NoOfStudents=@NewNoOfStudents
		WHERE ConfResID=@ConfResID
END
GO
```

CancelConfDayReservation – Anulowanie rezerwacji na dany dzień konferencji

```
CREATE PROCEDURE CancelConfDayReservation

	@ConfResID int
AS
BEGIN
	SET NOCOUNT ON;
	IF NOT EXISTS (SELECT * FROM ConferenceReservations WHERE ConfResID=@ConfResID)
	BEGIN
		RAISERROR ('No reservation with given ID',14,1)
		RETURN
	END
	UPDATE ConferenceReservations
		SET Cancelled=1
		WHERE ConfResID=@ConfResID
END
GO
```

CancelWorkshopReservation – Anulowanie rezerwacji na warsztat

```
CREATE PROCEDURE CancelWorkshopReservation

	@WorkResID int
AS
BEGIN
	SET NOCOUNT ON;
	IF NOT EXISTS(SELECT * FROM WorkshopReservations WHERE WorkResID=@WorkResID)
	BEGIN
		RAISERROR ('No reservation with given ID',14,1)
		RETURN
	END
	UPDATE WorkshopReservations
		SET Cancelled=1 WHERE WorkResID=@WorkResID
END
GO
```

CancelConfDayRegistration – Anulowanie rejestracji na dany dzień konferencji przez uczestnika

```
CREATE PROCEDURE CancelConfDayRegistration
	@RegistrationID int
AS
BEGIN
	SET NOCOUNT ON;
	IF NOT EXISTS (SELECT * FROM ConfDayRegistrations WHERE
	RegistrationID=@RegistrationID)
	BEGIN
		RAISERROR ('No conference day registration with given ID',14,1)
		RETURN
	END
	DELETE FROM WorkshopRegistrations
		WHERE RegistrationID IN (SELECT wreg.RegistrationID FROM
		WorkshopRegistrations wreg INNER JOIN WorkshopReservations wr ON
		wreg.ReservationID=wr.WorkResID INNER JOIN ConfDayRegistrations cdr ON
		wr.DayRegID=cdr.RegistrationID WHERE cdr.RegistrationID=@RegistrationID
		)

	DELETE FROM WorkshopReservations
		WHERE WorkResID IN (SELECT wr.WorkResID FROM WorkshopReservations wr INNER
		JOIN ConfDayRegistrations cdr ON wr.DayRegID=cdr.RegistrationID INNER JOIN
		ConferenceReservations cr ON cdr.RegistrationID=cr.ConfResID WHERE
		cdr.RegistrationID=@RegistrationID
		)
	DELETE FROM ConfDayRegistrationsWHERE RegistrationID=@RegistrationID
END
GO
```

FreeUnregisteredPlaces – Zmiana miejsc w rezerwacji na dzień konferencji na tyle, ilu jest zarejestrowanych uczestników

```
CREATE PROCEDURE FreeUnregisteredPlaces
      @ConfResID int
AS
BEGIN
      SET NOCOUNT ON;
      IF NOT EXISTS (SELECT * FROM ConferenceReservations WHERE ConfResID=@ConfResID)
      BEGIN
            RAISERROR ('No reservation with given ID',14,1)
            RETURN
      END
      DECLARE @Participants int
      SET @Participants=(SELECT COUNT(RegistrationID) FROM ConfDayRegistrations cdr
      WHERE cdr.ReservationID=@ConfResID)
      DECLARE @Students int
      SET @Students=(SELECT COUNT(p.ParticipantID)    FROM ConfDayRegistrations cdr
      INNER JOIN ConferenceReservations cr ON cdr.RegistrationID=cr.ConfResID
      INNER JOIN Participants p ON cdr.ParticipantID=p.ParticipantID WHERE
      p.Student=1 AND cr.ConfResID=@ConfResID)
      EXEC ChangeConferenceDayReservation
            @ConfResID,
            @Participants,
            @Students
END
GO
```

CancelUnpaidReservations – Anulowanie wszystkich rezerwacji nieopłaconych w ciągu 7 dni od zarezerwowania

```
CREATE PROCEDURE CancelUnpaidReservations
AS
BEGIN
      SET NOCOUNT ON;
      UPDATE ConferenceReservations
            SET Cancelled=1
            WHERE ConfResID IN (SELECT cr.ConfResID FROM Prices p INNER JOIN
            ConferenceDays cd ON p.DayID=cd.DayID INNER JOIN ConferenceReservations cr
            ON cd.DayID=cr.DayID LEFT OUTER JOIN Payments pay ON
            cr.ConfResID=pay.ConfResID WHERE (pay.PaymentID IS NULL OR
            pay.Paid<((cr.NoOfParticipants-cr.NoOfStudents)*(p.Price*(1-
            p.Discount))+cr.NoOfStudents*(p.Price*(1-p.Discount-p.StudentDiscount))))
            AND DATEDIFF(day,cr.ResDate,GETDATE())>7
                                    )
END
GO
```

EditConference – Edytowanie danych konferencji

```
CREATE PROCEDURE EditConference
	@ConferenceID int,
	@ConferenceName nvarchar(60),
	@Address nvarchar (50),
	@City nvarchar(40),
	@PostalCode nvarchar(10),
	@MaxPeople int
AS
BEGIN
	SET NOCOUNT ON;
	IF @ConferenceID IS NULL
	BEGIN
		RAISERROR ('ConferenceID is NULL. Can''t find conference',14,1)
		RETURN
	END
	IF @ConferenceName IS NOT NULL
	BEGIN
		UPDATE Conferences
		SET ConferenceName=@ConferenceName
		WHERE ConferenceID=@ConferenceID
	END
	IF @Address IS NOT NULL
	BEGIN
		UPDATE Conferences
		SET Address=@Address
		WHERE ConferenceID=@ConferenceID
	END
	IF @City IS NOT NULL
	BEGIN
		UPDATE Conferences
		SET City=@City
		WHERE ConferenceID=@ConferenceID
	END
	IF @PostalCode IS NOT NULL
	BEGIN
		UPDATE Conferences
		SET PostalCode=@PostalCode
		WHERE ConferenceID=@ConferenceID
	END
	IF @MaxPeople IS NOT NULL
	BEGIN
		UPDATE ConferenceDays
		SET MaxPeople=@MaxPeople
		WHERE DayID IN (SELECT cd.DayID FROM ConferenceDays cd INNER JOIN
		Conferences c ON cd.ConferenceID=c.ConferenceID
				WHERE c.ConferenceID=@ConferenceID)
	END
END
GO
```

EditClient – Edytowanie danych klienta

```
CREATE PROCEDURE EditClient
	@ClientID int,
	@Company bit,
	@CompanyName nvarchar(40),
	@LastName nvarchar(25),
	@FirstName nvarchar(25),
	@Address nvarchar(50),
	@City nvarchar(40),
	@PostalCode nvarchar(10),
	@Phone nvarchar(15)
AS
BEGIN
	SET NOCOUNT ON;
	IF @ClientID IS NULL
	BEGIN
		RAISERROR ('Client ID is NULL. Can''t find client',14,1)
		RETURN
	END
	IF @Company IS NOT NULL
	BEGIN
		UPDATE Clients
		SET Company=@Company WHERE ClientID=@ClientID
	END
	IF @CompanyName IS NOT NULL
	BEGIN
		UPDATE Clients
		SET CompanyName=@CompanyName WHERE ClientID=@ClientID AND
		Company=1
	END
	IF @LastName IS NOT NULL
	BEGIN
		UPDATE Clients
		SET LastName=@LastName WHERE ClientID=@ClientID
	END
	IF @FirstName IS NOT NULL
	BEGIN
		UPDATE Clients
		SET FirstName=@FirstName WHERE ClientID=@ClientID
	END
	IF @Address IS NOT NULL
	BEGIN
		UPDATE Clients
		SET Address=@Address WHERE ClientID=@ClientID
	END
	IF @City IS NOT NULL
	BEGIN
		UPDATE Clients
		SET City=@City WHERE ClientID=@ClientID
	END
	IF @PostalCode IS NOT NULL
	BEGIN
		UPDATE Clients
		SET PostalCode=@PostalCode WHERE ClientID=@ClientID
	END
	IF @Phone IS NOT NULL
	BEGIN
		UPDATE Clients
		SET Phone=@Phone WHERE ClientID=@ClientID
	END
END
GO
```

EditParticipant – Edytowanie danych uczestnika

```
CREATE PROCEDURE EditParticipant
      @ParticipantID int,
      @LastName nvarchar(25),
      @Firstname nvarchar(25),
      @Phone nvarchar(15),
      @Student bit,
      @StudentIDNo integer
AS
BEGIN
      SET NOCOUNT ON;
      IF @ParticipantID IS NULL
      BEGIN
            RAISERROR ('Participant ID is NULL. Can''t find client',14,1)
            RETURN
      END
      IF @LastName IS NOT NULL
      BEGIN
            UPDATE Participants
            SET LastName=@LastName
            WHERE ParticipantID=@ParticipantID
      END
      IF @FirstName IS NOT NULL
      BEGIN
            UPDATE Participants
            SET LastName=@LastName
            WHERE ParticipantID=@ParticipantID
      END
      IF @Phone IS NOT NULL
      BEGIN
            UPDATE Participants
            SET Phone=@Phone
            WHERE ParticipantID=@ParticipantID
      END
      IF @Student IS NOT NULL
      BEGIN
            UPDATE Participants
            SET Student=@Student
            WHERE ParticipantID=@ParticipantID
      END
      IF @StudentIDNo IS NOT NULL
      BEGIN
            UPDATE Participants
            SET StudentIDNo=@StudentIDNo
            WHERE ParticipantID=@ParticipantID AND Student=1
      END
END
GO
```

9. Funkcje

CalculatePayment – Zwraca obliczoną wartość, którą należy zapłacić za podaną rezerwację warsztatu lub konferencji (argument @Conference 0 lub 1)

```
CREATE FUNCTION CalculatePayment(
	@ReservationID int,
	@Conference int
	)
	RETURNS numeric(4,2)
AS
BEGIN
	DECLARE @Result numeric(4,2)
	IF @Conference=1
	BEGIN
		SET @Result=(SELECT((cr.NoOfParticipants-
		cr.NoOfStudents)*(p.Price*(1-
		p.Discount))+cr.NoOfStudents*(p.Price*(1-p.Discount-
		p.StudentDiscount)))
		FROM ConferenceReservations cr INNER JOIN
		ConferenceDays cd ON cr.DayID=cd.DayID AND
		cr.ConfResID=@ReservationID
		INNER JOIN Prices p ON p.DayID=cd.DayID WHERE
		cr.ConfResID=@ReservationID AND p.DaysTo=(SELECT TOP 1 pr.DaysTo
		FROM Prices pr WHERE pr.DayID=p.DayID AND
		DATEDIFF(day,cr.ResDate,cd.Date)>=pr.DaysTo ORDER BY pr.DaysTo)
				)
	END
	IF @Conference=0
	BEGIN
	IF EXISTS (SELECT * FROM WorkshopReservations wr INNER JOIN
	Workshops w ON wr.WorkshopID=w.WorkshopID
	INNER JOIN Prices p ON w.PriceID=p.PriceID WHERE
	wr.WorkResID=@ReservationID)
		BEGIN
			IF ((SELECT p.Student FROM Participants p INNER JOIN
			ConfDayRegistrations cdr ON
			p.ParticipantID=cdr.ParticipantID
			INNER JOIN WorkshopReservations wr ON
			cdr.RegistrationID=wr.DayRegID
			WHERE wr.WorkResID=@ReservationID)=1)
			BEGIN
				SET @Result=(SELECT p.Price*(1-p.Discount-
				p.StudentDiscount) FROM Prices p INNER JOIN
				Workshops w ON p.PriceID=w.PriceID INNER JOIN
				WorkshopReservations wr ON
				w.WorkshopID=wr.WorkshopID)
			END
			ELSE
			BEGIN
				SET @Result=(SELECT p.Price*(1-p.Discount) FROM
				Prices p INNER
				JOIN Workshops w ON p.PriceID=w.PriceID
				INNER JOIN WorkshopReservations wr ON
				w.WorkshopID=wr.WorkshopID)
			END
		END
	END
	RETURN @Result
END
GO
```

ConfDayAvailablePlaces – Zwraca ilość wolnych miejsc na podany dzień konferencji

```
CREATE FUNCTION ConfDayAvailablePlaces(
      @DayID int
      )
      RETURNS int
AS
BEGIN
      DECLARE @Result int
      SET @Result=(SELECT MaxPeople FROM ConferenceDays WHERE
      DayID=@DayID)-
                  (SELECT SUM(NoOfParticipants) FROM ConferenceReservations
                  WHERE
                  DayID=@DayID AND Cancelled=0)
      RETURN @Result
END
GO
```

WorkshopAvailablePlaces – Zwraca ilość wolnych miejsc na podany warsztat

```
CREATE FUNCTION WorkshopAvailablePlaces(
      @WorkshopID int
      )
      RETURNS int
AS
BEGIN
      DECLARE @Result int
            SET @Result=(SELECT MaxPeople FROM Workshops WHERE
                        WorkshopID=@WorkshopID)-
                  (SELECT COUNT(*) FROM WorkshopReservations WHERE
                  WorkshopID=@WorkshopID AND Cancelled=0)
      RETURN @Result
END
GO
```

10. Wnioski

Na koniec tego dokumentu chcielibyśmy przedstawić nasze wnioski, których dostrzeganie podczas całego etapu projektowania przyczyniło się do jeszcze lepszej optymalizacji naszej bazy danych.

Proces tworzenia bazy danych rozpoczęto od przedyskutowania oraz dokładnego przeanalizowania problemu stworzenia systemu do organizacji konferencji.

Nasza analiza zajęła nam bardzo dużo czasu ale co z tym idzie przyniosła ona dużo efektów, najważniejszym z nich było niewątpliwie to, że postanowiliśmy utworzyć indeksy na każdym kluczu obcym, ponieważ istniała możliwość że wartości pól będą rzadko się powtarzać, a dzięki zastosowaniu nieklastrowanych indeksów wyszukiwanie było szybsze.