Informatyka II rok, Podstawy baz danych
Baza"Konferencje"

Olga Słota
Damian Sosnowski

1. Wstęp

Realizowany projekt w ramach przedmiotu *Podstawy baz danych* miał na celu wprowadzenie w tematykę projektowania i implementacji systemów bazodanowych. Zadaniem do wykonania było stworzenie bazy danych dla firmy zajmującej się organizacją konferencji dla pracowników firm oraz osób prywatnych. Należało uwzględnić zasady rejestracji na konferencje i warsztaty, opłacania ich oraz odwoływania zawarte w instrukcji do zadania:

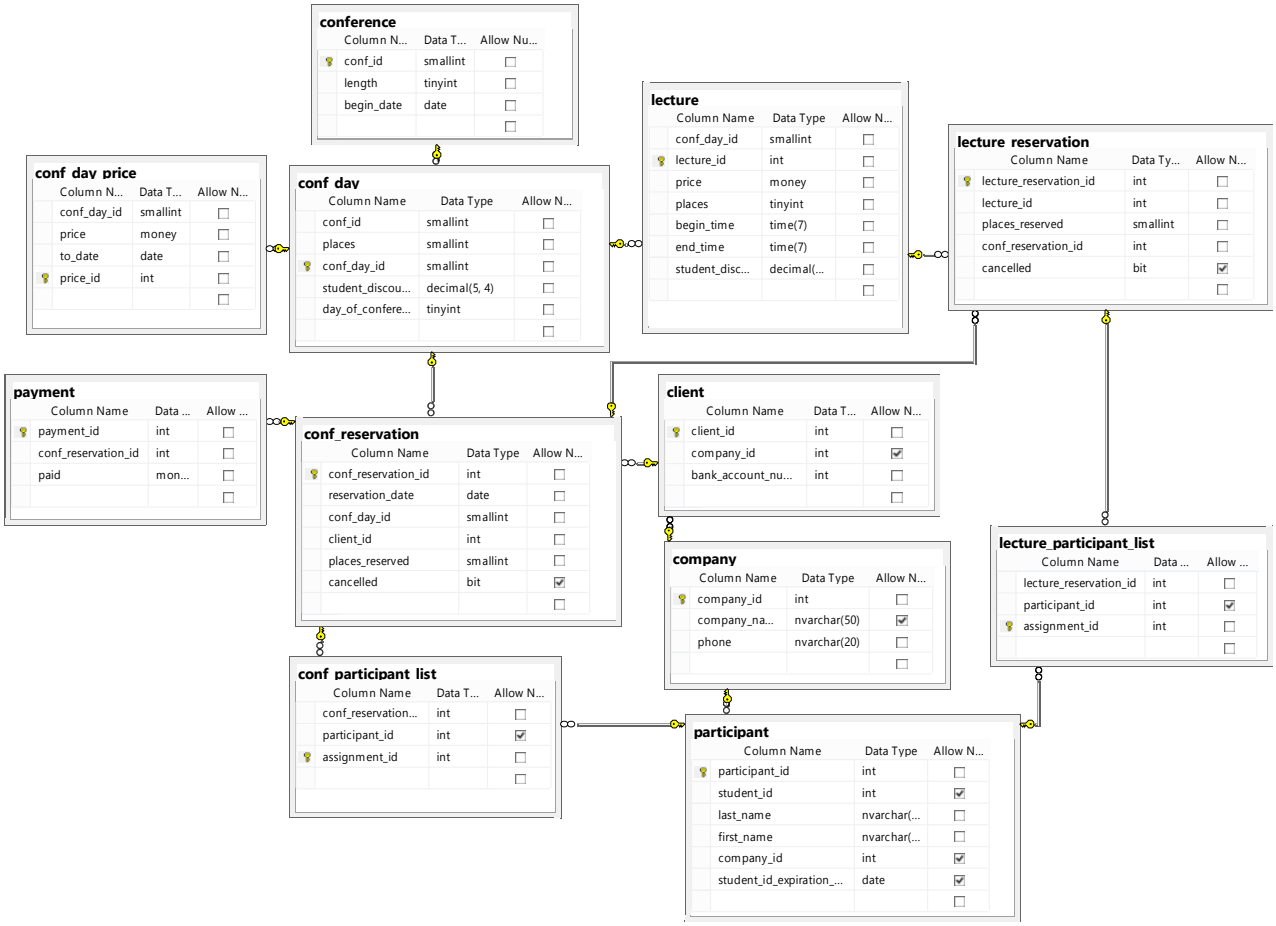
- klienci – firmy lub osoby prywatne
- konferencje – jedno lub kilkudniowe
- uczestnicy – osoby (firma może podać listę uczestników do 2 tygodni przed konferencją zamiast przy rezerwacji jak klient prywatny)
- warsztaty – rejestracja wymaga wcześniejszego zapisu na dzień konferencji
- student – przysługuje mu zniżka procentowa na konferencje i warsztaty pod warunkiem podania nr legitymacji przy rejestracji
- progi cenowe – zmieniają się wraz ze zbliżaniem się konferencji, dotyczą tylko dni konferencji , a nie warsztatów
- opłaty – maksymalnie tydzień od rejestracji w przeciwnym wypadku - anulowanie

Użytkownicy systemu bazodanowego:

- klient – może dokonać rezerwacji przez serwis www oraz w ciągu tygodnia opłacić ją
- uczestnik – bierze udział w konferencji /warsztacie , posiada identyfikator (imię, nazwisko, informacja o firmie)
- pracownik firmy konferencyjnej – wprowadza do systemu konferencje , określa ceny , odwołuje konferencje , ustala daty trwania i limity miejsc , ma dostęp do raportów o klientach, uczestnikach na dany dzień itd.

	Klient	Pracownik firmy konferencyjnej
procedury	• NewParticipant • ParticipateConf, • ParticipateLecture, • ConfReservation, • LectureReservation, • CancelConfReservation, • CancelLectureReservation • DelFromConfParticipantList, • DelFromLectureParticipantList • SetLecturePlacesNumber, • ReduceConfPlacesNumber	NewClient, NewCompany, NewConference, NewConfDay, NewParticipant, NewLecture, NewPayment, NewConfDayPrice, ParticipateConf, ParticipateLecture, ConfReservation, LectureReservation, CancelConfReservation, CancelLectureReservation DelFromConfParticipantList, DelFromLectureParticipantList SetLecturePlacesNumber, ReduceConfPlacesNumber
widoki	• LecturePayments • ConfDayPayments • ConfPayments • ConfDayParticipants • LectureParticipants • AvaliableLectures	• ClientsWithReservation • LecturePayments • ConfDayPayments • ConfPayments • BestClients • ConfDayParticipants • LectureParticipants • AvaliableLectures • ToPay • WeekAfterPartiallyPaidReservation

2. Diagram bazy danych :



3. Opis tabel :

Tabela conference - tabela opisująca konferencje

- conf_id – identyfikator – klucz główny
- length – długość w dniach
- begin_date - data rozpoczęcia

```
CREATE TABLE [dbo].[conference](
	[conf_id] [smallint] IDENTITY(1,1) NOT NULL,
	[length] [tinyint] NOT NULL,
	[begin_date] [date] NOT NULL,
 CONSTRAINT [PK_conference] PRIMARY KEY CLUSTERED
(
	[conf_id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO

ALTER TABLE [dbo].[conference]  WITH CHECK ADD  CONSTRAINT [CK_conference] CHECK
(([length]>=(1)))
GO

ALTER TABLE [dbo].[conference] CHECK CONSTRAINT [CK_conference]
GO
```

Tabela conf_day - tabela opisująca dzień konferencji

- conf_id – identyfikator konferencji – klucz obcy
- places – liczba miejsc na ten dzień konferencji
- conf_day_id – identyfikator – klucz główny
- student_discount – zniżka przysługująca studentom biorącym udział
- day_of_conference – numer dnia konferencji

```
CREATE TABLE [dbo].[conf_day](
	[conf_id] [smallint] NOT NULL,
	[places] [smallint] NOT NULL,
	[conf_day_id] [smallint] IDENTITY(1,1) NOT NULL,
	[student_discount] [decimal](5, 4) NOT NULL,
	[day_of_conference] [tinyint] NOT NULL,
 CONSTRAINT [PK_conf_day_1] PRIMARY KEY CLUSTERED
(
	[conf_day_id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO

ALTER TABLE [dbo].[conf_day]  WITH CHECK ADD  CONSTRAINT [FK_conf_day_conference]
FOREIGN KEY([conf_id])
REFERENCES [dbo].[conference] ([conf_id])
GO

ALTER TABLE [dbo].[conf_day] CHECK CONSTRAINT [FK_conf_day_conference]
GO

ALTER TABLE [dbo].[conf_day]  WITH CHECK ADD  CONSTRAINT [CK_conf_day] CHECK
(([student_discount]<=(1)))
GO

ALTER TABLE [dbo].[conf_day] CHECK CONSTRAINT [CK_conf_day]
GO

ALTER TABLE [dbo].[conf_day]  WITH CHECK ADD  CONSTRAINT [CK_conf_day_1] CHECK
(([day_of_conference]>=(1)))
GO

ALTER TABLE [dbo].[conf_day] CHECK CONSTRAINT [CK_conf_day_1]
GO

ALTER TABLE [dbo].[conf_day]  WITH CHECK ADD  CONSTRAINT [CK_conf_day_2] CHECK
(([places]>=(1)))
GO

ALTER TABLE [dbo].[conf_day] CHECK CONSTRAINT [CK_conf_day_2]
GO

ALTER TABLE [dbo].[conf_day]  WITH CHECK ADD  CONSTRAINT [CK_conf_day_3] CHECK
(([student_discount]>=(0)))
GO

ALTER TABLE [dbo].[conf_day] CHECK CONSTRAINT [CK_conf_day_3]
GO
```

Tabela lecture - tabela opisująca warsztaty

- conf_day_id – identyfikator dnia konferencji – klucz obcy
- lecture_id – identyfikator – klucz główny
- price – cena za warsztat
- places – liczba miejsc przewidzianych na warsztat
- begin_time – godzina rozpoczęcia
- end_time – godzina zakończenia
- student_discount – zniżka przysługująca studentom biorącym udział

```
CREATE TABLE [dbo].[lecture](
	[conf_day_id] [smallint] NOT NULL,
	[lecture_id] [int] IDENTITY(1,1) NOT NULL,
	[price] [money] NOT NULL,
	[places] [tinyint] NOT NULL,
	[begin_time] [time](7) NOT NULL,
	[end_time] [time](7) NOT NULL,
	[student_discount] [decimal](5, 4) NOT NULL,
 CONSTRAINT [PK_lecture_1] PRIMARY KEY CLUSTERED
(
	[lecture_id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO

ALTER TABLE [dbo].[lecture]  WITH CHECK ADD  CONSTRAINT [FK_lecture_conf_day] FOREIGN
KEY([conf_day_id])
REFERENCES [dbo].[conf_day] ([conf_day_id])
GO

ALTER TABLE [dbo].[lecture] CHECK CONSTRAINT [FK_lecture_conf_day]
GO

ALTER TABLE [dbo].[conf_reservation]  WITH CHECK ADD  CONSTRAINT [CK_conf_reservation]
CHECK  (([places_reserved]>=(1)))
GO

ALTER TABLE [dbo].[conf_reservation] CHECK CONSTRAINT [CK_conf_reservation]
GO

ALTER TABLE [dbo].[lecture]  WITH CHECK ADD  CONSTRAINT [CK_lecture] CHECK
(([places]>=(0)))
GO

ALTER TABLE [dbo].[lecture] CHECK CONSTRAINT [CK_lecture]
GO

ALTER TABLE [dbo].[lecture]  WITH CHECK ADD  CONSTRAINT [CK_lecture_1] CHECK
(([student_discount]>=(0)))
GO

ALTER TABLE [dbo].[lecture] CHECK CONSTRAINT [CK_lecture_1]
GO

ALTER TABLE [dbo].[lecture]  WITH CHECK ADD  CONSTRAINT [CK_lecture_2] CHECK
(([student_discount]<=(1)))
GO

ALTER TABLE [dbo].[lecture] CHECK CONSTRAINT [CK_lecture_2]
```

Tabela client - tabela opisująca klienta

- client_id - identyfikator – klucz główny
- company_id – identyfikator firmy (jeśli klient nie jest firmą – null)
- bank_account_number – nr konta klienta

```
CREATE TABLE [dbo].[client](
	[client_id] [int] IDENTITY(1,1) NOT NULL,
	[company_id] [int] NULL,
	[bank_account_number] [nvarchar](40) NOT NULL,
 CONSTRAINT [PK_client] PRIMARY KEY CLUSTERED
(
	[client_id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO

ALTER TABLE [dbo].[client]  WITH CHECK ADD  CONSTRAINT [FK_client_company] FOREIGN
KEY([company_id])
REFERENCES [dbo].[company] ([company_id])
GO

ALTER TABLE [dbo].[client] CHECK CONSTRAINT [FK_client_company]
GO
```

Tabela company - tabela opisująca firmę będącą klientem

- company_id - identyfikator – klucz główny
- company_name– nazwa firmy
- phone – nr telefonu do firmy

```
CREATE TABLE [dbo].[company](
	[company_id] [int] IDENTITY(1,1) NOT NULL,
	[company_name] [nvarchar](50) NULL,
	[phone] [nvarchar](20) NOT NULL,
 CONSTRAINT [PK_company] PRIMARY KEY CLUSTERED
(
	[company_id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO
```

Tabela participant - tabela opisująca uczestnika

- participant_id - identyfikator – klucz główny
- sudent_id – nr legitymacji (null – jeśli brak)
- last_name - nazwisko
- first_name – imię
- company_id – identyfikator firmy (jeśli uczestnik nie jest pracownikiem firmy – null)
- student_id_expiration_date – data ważności legitymacji (null – jeśli brak legitymacji)

```
CREATE TABLE [dbo].[participant](
	[participant_id] [int] IDENTITY(1,1) NOT NULL,
	[student_id] [int] NULL,
	[last_name] [nvarchar](20) NOT NULL,
	[first_name] [nvarchar](20) NOT NULL,
	[company_id] [int] NULL,
	[student_id_expiration_date] [date] NULL,
 CONSTRAINT [PK_person_1] PRIMARY KEY CLUSTERED
(
	[participant_id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO

ALTER TABLE [dbo].[participant]  WITH CHECK ADD  CONSTRAINT [FK_person_company]
FOREIGN KEY([company_id])
REFERENCES [dbo].[company] ([company_id])
GO

ALTER TABLE [dbo].[participant] CHECK CONSTRAINT [FK_person_company]
GO
```

Tabela payment - tabela opisująca opłaty

- payment_id - identyfikator – klucz główny
- conf_reservation_id –identyfikator opłacanej rezerwacji – klucz obcy
- paid – wpłacona kwota

```
CREATE TABLE [dbo].[payment](
	[payment_id] [int] IDENTITY(1,1) NOT NULL,
	[conf_reservation_id] [int] NOT NULL,
	[paid] [money] NOT NULL,
 CONSTRAINT [PK_payment] PRIMARY KEY CLUSTERED
(
	[payment_id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO

ALTER TABLE [dbo].[payment]  WITH CHECK ADD  CONSTRAINT [FK_payment_conf_reservation]
FOREIGN KEY([conf_reservation_id])
REFERENCES [dbo].[conf_reservation] ([conf_reservation_id])
GO

ALTER TABLE [dbo].[payment] CHECK CONSTRAINT [FK_payment_conf_reservation]
GO
```

Tabela conf_day_price - tabela opisująca progi cenowe dni konferencji

- conf_day_id - identyfikator dnia konferencji– klucz obcy
- price – obowiązująca cena
- to_date – data wygaśnięcia danego progu cenowego
- price_id – identyfikator – klucz główny

```
CREATE TABLE [dbo].[conf_day_price](
	[conf_day_id] [smallint] NOT NULL,
	[price] [money] NOT NULL,
	[to_date] [date] NOT NULL,
	[price_id] [int] IDENTITY(1,1) NOT NULL,
 CONSTRAINT [PK_conf_day_price] PRIMARY KEY CLUSTERED
(
	[price_id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO

ALTER TABLE [dbo].[conf_day_price]  WITH CHECK ADD  CONSTRAINT [FK_discount_conf_day]
FOREIGN KEY([conf_day_id])
REFERENCES [dbo].[conf_day] ([conf_day_id])
GO

ALTER TABLE [dbo].[conf_day_price] CHECK CONSTRAINT [FK_discount_conf_day]
GO
```

Tabela conf_reservation - tabela opisująca rezerwacje na dzień konferencji

- conf_reservation_id – identyfikator rezerwacji - klucz główny
- reservation_date – data dokonania rezerwacji
- conf_day_id – identyfikator dnia– klucz obcy
- client_id – identyfikator klienta – klucz obcy
- places_reserved – liczba zarezerwowanych miejsc
- cancelled – 1 - jeśli rezerwacja odwołana , 0 lub null - jeśli nie

```
CREATE TABLE [dbo].[conf_reservation](
	[conf_reservation_id] [int] IDENTITY(1,1) NOT NULL,
	[reservation_date] [date] NOT NULL,
	[conf_day_id] [smallint] NOT NULL,
	[client_id] [int] NOT NULL,
	[places_reserved] [smallint] NOT NULL,
	[cancelled] [bit] NULL,
 CONSTRAINT [PK_conf_reservation] PRIMARY KEY CLUSTERED
(
	[conf_reservation_id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO

ALTER TABLE [dbo].[conf_reservation]  WITH CHECK ADD  CONSTRAINT
[FK_conf_reservation_client] FOREIGN KEY([client_id])
REFERENCES [dbo].[client] ([client_id])
GO

ALTER TABLE [dbo].[conf_reservation] CHECK CONSTRAINT [FK_conf_reservation_client]
GO

ALTER TABLE [dbo].[conf_reservation]  WITH CHECK ADD  CONSTRAINT
[FK_conf_reservation_conf_day] FOREIGN KEY([conf_day_id])
REFERENCES [dbo].[conf_day] ([conf_day_id])
GO

ALTER TABLE [dbo].[conf_reservation] CHECK CONSTRAINT [FK_conf_reservation_conf_day]
GO
```

Tabela lecture_reservation - tabela opisująca rezerwacje na warsztat

- lecture_reservation_id – identyfikator rezerwacji - klucz główny
- conf_reservation_id – identyfikator rezerwacji tego dnia konferencji – klucz obcy
- lecture_id – identyfikator warsztatu– klucz obcy
- places_reserved – liczba zarezerwowanych miejsc
- cancelled – 1 - jeśli rezerwacja odwołana , 0 lub null - jeśli nie

```
CREATE TABLE [dbo].[lecture_reservation](
	[lecture_reservation_id] [int] IDENTITY(1,1) NOT NULL,
	[lecture_id] [int] NOT NULL,
	[places_reserved] [smallint] NOT NULL,
	[conf_reservation_id] [int] NOT NULL,
	[cancelled] [bit] NULL,
 CONSTRAINT [PK_lecture_reservation] PRIMARY KEY CLUSTERED
(
	[lecture_reservation_id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO

ALTER TABLE [dbo].[lecture_reservation]  WITH CHECK ADD  CONSTRAINT
[FK_lecture_reservation_conf_reservation] FOREIGN KEY([conf_reservation_id])
REFERENCES [dbo].[conf_reservation] ([conf_reservation_id])
GO

ALTER TABLE [dbo].[lecture_reservation] CHECK CONSTRAINT
[FK_lecture_reservation_conf_reservation]
GO

ALTER TABLE [dbo].[lecture_reservation]  WITH CHECK ADD  CONSTRAINT
[FK_lecture_reservation_lecture] FOREIGN KEY([lecture_id])
REFERENCES [dbo].[lecture] ([lecture_id])
GO

ALTER TABLE [dbo].[lecture_reservation] CHECK CONSTRAINT
[FK_lecture_reservation_lecture]
GO

ALTER TABLE [dbo].[lecture_reservation]  WITH CHECK ADD  CONSTRAINT
[CK_lecture_reservation] CHECK  (([places_reserved]>=(1)))
GO

ALTER TABLE [dbo].[lecture_reservation] CHECK CONSTRAINT [CK_lecture_reservation]
GO
```

Tabela conf_participant_list - tabela opisująca rezerwacje dokonane dla poszczególnych uczestników tj. przypisanie uczestnika do rezerwacji dokonanej przez klienta i wpisanie go na listę uczestników

- conf_reservation_id – identyfikator rezerwacji - klucz obcy
- participant_id – identyfikator klienta – klucz obcy
- assignment_id – identyfikator przypisania uczestnika do dnia konferencji – klucz główny

```
CREATE TABLE [dbo].[conf_participant_list](
	[conf_reservation_id] [int] NOT NULL,
	[participant_id] [int] NULL,
	[assignment_id] [int] IDENTITY(1,1) NOT NULL,
 CONSTRAINT [PK_conf_participant_list] PRIMARY KEY CLUSTERED
(
	[assignment_id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO

ALTER TABLE [dbo].[conf_participant_list]  WITH CHECK ADD  CONSTRAINT
[FK_conf_participant_list_conf_reservation] FOREIGN KEY([conf_reservation_id])
REFERENCES [dbo].[conf_reservation] ([conf_reservation_id])
GO

ALTER TABLE [dbo].[conf_participant_list] CHECK CONSTRAINT
[FK_conf_participant_list_conf_reservation]
GO

ALTER TABLE [dbo].[conf_participant_list]  WITH CHECK ADD  CONSTRAINT
[FK_conf_participant_list_participant] FOREIGN KEY([participant_id])
REFERENCES [dbo].[participant] ([participant_id])
GO

ALTER TABLE [dbo].[conf_participant_list] CHECK CONSTRAINT
[FK_conf_participant_list_participant]
GO
```

Tabela lecture_participant_list - tabela opisująca rezerwacje dokonane dla poszczególnych uczestników tj. przypisanie uczestnika do rezerwacji dokonanej przez klienta i wpisanie go na listę uczestników

- lecture_reservation_id – identyfikator rezerwacji - klucz obcy
- participant_id – identyfikator klienta – klucz obcy
- assignment_id – identyfikator przypisania uczestnika do warsztatu – klucz główny

```
CREATE TABLE [dbo].[lecture_participant_list](
	[lecture_reservation_id] [int] NOT NULL,
	[participant_id] [int] NULL,
	[assignment_id] [int] IDENTITY(1,1) NOT NULL,
 CONSTRAINT [PK_lecture_participant_list] PRIMARY KEY CLUSTERED
(
	[assignment_id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO

ALTER TABLE [dbo].[lecture_participant_list]  WITH CHECK ADD  CONSTRAINT
[FK_lecture_participant_list_lecture_reservation] FOREIGN KEY([lecture_reservation_id])
REFERENCES [dbo].[lecture_reservation] ([lecture_reservation_id])
GO

ALTER TABLE [dbo].[lecture_participant_list] CHECK CONSTRAINT
[FK_lecture_participant_list_lecture_reservation]
GO

ALTER TABLE [dbo].[lecture_participant_list]  WITH CHECK ADD  CONSTRAINT
[FK_lecture_participant_list_participant] FOREIGN KEY([participant_id])
REFERENCES [dbo].[participant] ([participant_id])
GO

ALTER TABLE [dbo].[lecture_participant_list] CHECK CONSTRAINT
[FK_lecture_participant_list_participant]
GO
```

4. Widoki :

ClientsWithReservation - widok pokazujący klientów i liczbę miejsc zarezerwowanych dla ich rezerwacji oraz liczbą uczestników już do nich przypisanych.

```
CREATE VIEW ClientsWithReservation AS

SELECT client.client_id as [client id] , company.company_name as name , company.phone
as phone,
conf_reservation.places_reserved, conf_reservation.reservation_date ,
(select COUNT(assignment_id) from conf_participant_list
where conf_participant_list.conf_reservation_id = conf_reservation.conf_reservation_id)
as [places assigned]
FROM client join conf_reservation
on client.client_id = conf_reservation.client_id
inner join company
on company.company_id = client.company_id
```

LecturePayments - widok prezentujący informację o miejscach zarezerwowanych na warsztaty, liczbie osób do nich przypisanych oraz cenie jaką trzeba za nie zapłacić w danym momencie oraz maksymalną jaką trzeba by było zapłacić (bez rabatu studenckiego na miejsca zarezerwowane, ale nie przypisane).

```
CREATE VIEW LecturePayments AS
SELECT lr.lecture_reservation_id as "Lecture reservation id", lr.places_reserved as
"Places reserved",
count(lpl.assignment_id) as "Places assigned", l.price as "Price for normal place",
l.student_discount as "student discount", count(p.student_id) as "Students included",
(l.price*lr.places_reserved) as "Price at the most",
(cast((l.price*((count(lpl.assignment_id)-count(p.student_id))+(count(p.student_id))*(1-
(l.student_discount))))as numeric(10,2)))
 as "Price for places reserved"

FROM lecture_reservation as lr
INNER JOIN lecture as l
ON lr.lecture_id=l.lecture_id
INNER JOIN conf_day as cd
ON l.conf_day_id=cd.conf_day_id
INNER JOIN conference as c
ON cd.conf_id=c.conf_id
LEFT OUTER JOIN lecture_participant_list as lpl
ON lr.lecture_reservation_id=lpl.lecture_reservation_id
LEFT OUTER JOIN participant as p
ON (lpl.participant_id=p.participant_id AND
DATEDIFF(day, c.begin_date, p.student_id_expiration_date)>=0)
GROUP BY lr.lecture_reservation_id, lr.places_reserved,l.price,l.student_discount
```

ConfDayPayments - widok, który przedstawia informacje na temat danego dnia konferencji. Podaje liczbę miejsc zarezerwowanych, uczestników już przypisanych, studentów, a także cenę jaką muszą zapłacić klienci za konferencję w danej chwili oraz maksymalną jaką trzeba by było zapłacić (bez rabatu studenckiego na miejsca zarezerwowane, ale nie przypisane).

```
CREATE VIEW ConfDayPayments AS

SELECT c.conf_id, cr.conf_reservation_id , cr.conf_day_id ,cr.client_id
,cr.reservation_date ,
  cr.places_reserved as [places reserved], count(assignment_id) as [places assigned],
  count (p.student_id) as students , (CAST((pr.price *((count(cr.conf_reservation_id) -
count(p.student_id))
  +count(p.student_id)*cd.student_discount)) as numeric(10,2))) as 'Conference day act
price',
  (dbo.GetPriceStageForDate(cr.reservation_date,cr.conf_day_id) * cr.places_reserved) as
'Conference day max price',
  (isnull(WI.act_price,0)) as 'Lectures act price',
  isnull((WI.up_price),0) as 'Lectures max price'

FROM conf_reservation as cr
join conf_day as cd
on cr.conf_day_id = cd.conf_day_id
join conference as c
on c.conf_id = cd.conf_id
left join conf_participant_list as cpl
on cpl.conf_reservation_id = cr.conf_reservation_id
left join participant as p
on p.participant_id=cpl.participant_id
AND(DATEDIFF(day,c.begin_date,p.student_id_expiration_date)>=0)
join conf_day_price as pr
on pr.price_id = (dbo.GetConfPriceID(cr.reservation_date,cr.conf_day_id))
left join (SELECT [Lecture reservation id] as conf_day_reservation, sum([Price for places
reserved]) as act_price,
sum([Price at the most]) as up_price
FROM LecturePayments
GROUP BY [Lecture reservation id]) as WI
ON cr.conf_reservation_id = WI.conf_day_reservation
GROUP BY cr.conf_reservation_id, cr.conf_day_id, cr.client_id, cr.reservation_date,
cr.places_reserved,
pr.price, cd.student_discount, c.conf_id, WI.act_price, WI.up_price
```

ConfPayments - widok liczący cenę rezerwacji całej konferencji, wszystkich jej dni, dzieląc na klientów i dane konferencje.

```
CREATE VIEW ConfPayments AS
SELECT client_id, conf_id,
cast((sum([Conference day act price])+sum([Lectures act price])) as numeric(10,2))
as 'Act price to pay for conference',
cast((sum([Conference day max price])+sum([Lectures max price])) as numeric(10,2))
as 'Max price to pay for conference'

FROM ConfDayPayments
GROUP BY client_id, conf_id
```

AvailableLectures - widok przedstawiający warsztaty, które mają miejsca wolne. Ponadto pokazuje ich podstawowe dane.

```
CREATE VIEW AvailableLectures AS
SELECT cd.conf_day_id as "Conference Day ID", l.lecture_id as "Lecture ID",
dbo.GetLecturePlacesLeft(lecture_id) as "Places left", l.price as "Price",
l.student_discount*100 as "Student discount in percents", (DATEADD(day, 1, c.begin_date))
as "Lecture date",
l.begin_time as "Begin time", l.end_time as "End time"
FROM lecture as l
INNER JOIN conf_day as cd
ON l.conf_day_id=cd.conf_day_id
INNER JOIN conference as c
ON cd.conf_id=c.conf_id
WHERE (dbo.GetLecturePlacesLeft(lecture_id)>0)
```

BestCustomers - *widok przedstawiający klientów, którzy najczęściej wykonują rezerwacje na konferencje oraz najwięcej płacą.*

```
CREATE VIEW BestCustomers AS
SELECT c.client_id as "Client ID", c.bank_account_number as "Client Account Number",
c.company_id as "Company ID if applicable", com.company_name as "Company name",
com.phone as "Company phone number", SUM(cr.places_reserved) AS "Total places reserved",
SUM(p.paid) AS "Totalpayments", COUNT (DISTINCT cd.conf_id) AS "Conferences participated"

FROM client as c
INNER JOIN conf_reservation as cr
ON c.client_id=cr.client_id
INNER JOIN payment as p
ON cr.conf_reservation_id=p.conf_reservation_id
INNER JOIN conf_day as cd
ON cr.conf_day_id=cd.conf_day_id
INNER JOIN company as com
ON c.company_id=com.company_id
GROUP BY c.client_id,c.bank_account_number,c.company_id,company_name,phone
```

ConfDayParticipants - widok przedstawiający uczestników na dany dzień konferencji włącznie z id klienta, który ich tam wpisał.

```
CREATE VIEW ConfDayParticipants AS
SELECT cr.conf_day_id, p.participant_id, p.first_name, p.last_name, cr.client_id

FROM conf_reservation as cr
INNER JOIN conf_participant_list as cpl
ON cr.conf_reservation_id=cpl.conf_reservation_id
INNER JOIN participant as p
ON cpl.participant_id=p.participant_id
GROUP BY cr.conf_day_id,p.participant_id,first_name,last_name,cr.client_id
```

LectureParticipants - widok przedstawiający uczestników na dany warsztat włącznie z id klienta, który ich tam wpisał.

```
CREATE VIEW LectureParticipants AS

SELECT lr.lecture_id, p.participant_id, p.first_name, p.last_name, cr.client_id
FROM lecture_reservation as lr
INNER JOIN lecture_participant_list as lpl
ON lr.lecture_reservation_id=lpl.lecture_reservation_id
INNER JOIN participant as p
ON lpl.participant_id=p.participant_id
INNER JOIN conf_reservation as cr
ON cr.conf_reservation_id=lr.conf_reservation_id
GROUP BY lr.lecture_id,p.participant_id, p.first_name, p.last_name, cr.client_id
```

ToPay - widok przedstawiający, ile klient zapłacił za daną konferencję, a ile powinien był.

```
CREATE VIEW ToPay AS
SELECT p.conf_reservation_id as "Conference day reservation id", cast(SUM(p.paid)
as numeric(10,2)) as "Paid money",
 cast(cb.[Conference day act price]+cb.[Lectures act price] as numeric(10,2))
as "Act price to pay for reservation",
cast(cb.[Conference day max price]+cb.[Lectures max price] as numeric(10,2))
as "Max price to pay for reservation"

FROM payment as p
INNER JOIN ConfDayPayments as cb
ON p.conf_reservation_id=cb.conf_reservation_id
GROUP BY (cb.[Conference day act price]+cb.[Lectures act price]),
(cb.[Conference day max price]+cb.[Lectures max price]), p.conf_reservation_id
```

WeekAfterPartiallyPaidReservation - widok przedstawiający listę klientów do których należy zadzwonić, ponieważ po tygodniu od złożenia rezerwacji wpłacili jedynie minimalną część kwoty, a nie jej całość, gdzie minimalna kwota to cena za przypisanych już uczestników.

```
CREATE VIEW WeekAfterPartialyPaidReservation AS
SELECT CI.conf_reservation_id, CI.reservation_date, c.conf_id, c.begin_date, CI.[places
reserved], CI.[Places assigned],
  isnull(PAI.[Paid money],0) as 'Paid money', CI.client_id, company.company_name as
'Client name',
  company.phone as 'Client phone',ISNULL( CL.company_id, 'no') as 'Is company'
FROM ConfDayPayments as CI
LEFT OUTER JOIN ToPay as PAI
ON (CI.conf_reservation_id = PAI.[Conference day reservation id])AND(PAI.[Paid money] >
(CI.[Conference day act price]+CI.[Lectures act price]))
INNER JOIN conf_reservation as cr ON (CI.conf_reservation_id =
cr.conf_reservation_id)AND((cr.cancelled = 0))
INNER JOIN conf_day as cd
ON CI.conf_day_id = cd.conf_day_id
INNER JOIN conference as c
ON cd.conf_id = c.conf_id
INNER JOIN Client as CL
ON CI.client_id = CL.client_id
LEFT JOIN company
on company.company_id = CL.company_id
 --Date 14 or less days before conference is starting, but still before conference
 WHERE ((DATEDIFF(day, GETDATE(), c.begin_date) <= 14) AND (DATEDIFF(day, GETDATE(),
c.begin_date) >= 0))
```

5. Funkcje :

GetConfPlacesLeft - funkcja, która zwraca ilość miejsc wolnych danego dnia konferencji.

```
CREATE FUNCTION [dbo].[GetConfPlacesLeft]
( @ConferenceDayId smallint )
	RETURNS smallint
AS
BEGIN
	IF not exists (SELECT conf_day.conf_day_id
			FROM conf_day
				WHERE conf_day.conf_day_id = @ConferenceDayId)
	  BEGIN
	RETURN 0
	  END

	ELSE
	  BEGIN

	DECLARE @placesLeft smallint
	DECLARE curs CURSOR LOCAL FOR
		SELECT places_reserved
		FROM conf_reservation
	    WHERE conf_day_id = @ConferenceDayId
	    DECLARE @reserved smallint

		-- Get all places
	    SET @placesLeft = (SELECT places FROM conf_day
				WHERE conf_day_id = @ConferenceDayId)

		-- booked places
	    SET @reserved = (SELECT sum(places_reserved) FROM conf_reservation
				WHERE (conf_day_id = @ConferenceDayId) AND
(isnull(cancelled,0) = 0)
				GROUP BY conf_day_id )

	IF(@reserved is not null)
		SET @placesLeft -= @reserved
	  END

	RETURN @placesLeft
END
```

GetLecturePlacesLeft - funkcja, która zwraca ilość miejsc wolnych na dany warsztat.

```
CREATE FUNCTION [dbo].[GetLecturePlacesLeft]
(
	@LectureId int
)
RETURNS tinyint
AS
BEGIN
	IF not exists (SELECT lecture_id
				FROM lecture
				WHERE lecture_id=@LectureId)
	BEGIN
		RETURN 0
	END
	ELSE
	BEGIN
		-- Get whole amount of free places
		DECLARE @placesLeft tinyint = (SELECT places FROM lecture
									WHERE lecture_id=@LectureId)
		DECLARE curs CURSOR LOCAL FOR
				SELECT places_reserved
				FROM lecture_reservation
				WHERE lecture_id=@LectureId
		DECLARE @reserved tinyint

		-- Substract booked places
		SET @reserved = (SELECT sum(places_reserved)
							FROM lecture_reservation
							WHERE (lecture_id=@LectureId) AND
							(isnull(cancelled,0)=0)
							GROUP BY lecture_id)
		IF(@reserved is not null)
			SET @placesLeft -= @reserved
		END
	-- Return the result of the function
	RETURN @placesLeft
END
```

GetConfPriceID - funkcja zwracająca klucz główny tabeli conf_day_price odpowiadający podanej dacie i dniu konferencji.

```
CREATE FUNCTION [dbo].[GetConfPriceID]
(
 @Date date,
 @ConferenceDayId smallint
)
RETURNS int

AS
BEGIN

  DECLARE @PriceId int = (SELECT TOP 1 price_id FROM conf_day_price
                  WHERE (conf_day_id = @ConferenceDayId) AND
(DATEDIFF(day,@date,to_date) >= 0)
                    ORDER BY to_date)

  RETURN @PriceId
END
```

GetConfPriceID - funkcja zwracająca klucz główny tabeli conf_day_price odpowiadający podanej dacie i dniu konferencji.

```
CREATE FUNCTION [dbo].[GetPriceStageForDate]
(
 @Date date,
 @ConferenceDayId smallint
)
RETURNS money

AS
BEGIN

  DECLARE @Price money = (SELECT TOP 1 price FROM conf_day_price
                     WHERE (conf_day_id = @ConferenceDayId) AND
(DATEDIFF(day,@date,to_date) >= 0)
                          ORDER BY to_date)

       RETURN @Price
END
```

6. Procedury :

NewClient – procedura dodająca nowego klienta

```
CREATE PROCEDURE [dbo].[NewClient]
 @company int,
 @account int
 AS
   BEGIN
     SET NOCOUNT ON;
     INSERT INTO client( company_id , bank_account_number)
     VALUES(@company, @account)
   END
```

NewCompany– procedura dodająca nową firmę będącą klientem

```
CREATE PROCEDURE [dbo].[NewCompany]
  @name int,
  @phone int
  AS
   BEGIN
       SET NOCOUNT ON;
       INSERT INTO company(company_name , phone)
       VALUES(@name, @phone)
   END
```

NewConference – procedura dodająca nową konferencję

```
CREATE PROCEDURE [dbo].[NewConference]
 @len tinyint,
 @begin date
 AS
 BEGIN
       SET NOCOUNT ON;
       INSERT INTO conference( length , begin_date)
       VALUES(@len, @begin)
 END
```

NewConfDay – procedura dodająca nowy dzień konferencji

```
CREATE PROCEDURE [dbo].[NewConfDay]
 @conf_id smallint,
 @places smallint,
 @discount decimal(5, 2),
 @day_no tinyint
 AS
   BEGIN
       SET NOCOUNT ON;
       INSERT INTO conf_day(conf_id,places, student_discount, day_of_conference)
       VALUES(@conf_id,@places, @discount, @day_no )
   END
```

NewParticipant – procedura dodająca nowego uczestnika

```
CREATE PROCEDURE [dbo].[NewParticipant]
  @student_id int,
  @last nvarchar(20),
  @first nvarchar(20),
  @company int,
  @student_date date
  AS
    BEGIN
       SET NOCOUNT ON;
       INSERT INTO participant(student_id , last_name , first_name , company_id,
student_id_expiration_date)
       VALUES(@student_id, @last , @first,@company, @student_date)
    END
```

NewLecture – procedura dodająca nowy warsztat

```
CREATE PROCEDURE [dbo].[NewLecture]
 @day_id smallint,
 @pr money,
 @pl tinyint,
 @begin time(7),
 @end time(7),
 @student decimal(5,2)
 AS
   BEGIN
      SET NOCOUNT ON;
      INSERT INTO lecture(conf_day_id, price,places, begin_time, end_time,student_discount)
      VALUES(@day_id, @pr,@pl , @begin , @end, @student)
   END
```

NewPayment – procedura dodająca nową opłatę

```
CREATE PROCEDURE [dbo].[NewPayment]
 @reservation_id int,
 @paid money
 AS
   BEGIN
      SET NOCOUNT ON;
      INSERT INTO payment(conf_reservation_id , paid)
      VALUES( @reservation_id,@paid)
   END
```

NewConfDayPrice – procedura dodająca nowy próg cenowy obowiązujący przez dany okres

```
CREATE PROCEDURE [dbo].[NewConfDayPrice]
 @day_id int,
 @pr money,
 @to date
 AS
   BEGIN
      SET NOCOUNT ON;
      INSERT INTO conf_day_price(conf_day_id,price,to_date)
      VALUES(@day_id, @pr, @to)
   END
```

ParticipateConf – procedura wpisująca uczestnika na listę z danego dnia konferencji

```
CREATE PROCEDURE [dbo].[ParticipateConf]
 @reservation_id int,
 @paticip_id int
 AS
   BEGIN
      SET NOCOUNT ON;
      INSERT INTO conf_participant_list(conf_reservation_id,participant_id)
      VALUES(@reservation_id,@paticip_id)
   END
```

ParticipateLecture – procedura wpisująca uczestnika na listę z danego warsztatu

```
CREATE PROCEDURE [dbo].[ParticipateLecture]
 @reservation_id int,
 @particip_id int
 AS
  BEGIN
    SET NOCOUNT ON;

    DECLARE @ConferenceDayReservationId int = (SELECT conf_reservation_id
                                               FROM lecture_reservation
                                               WHERE lecture_reservation_id =
                                               @reservation_id)

    DECLARE @AssignmentId int = (SELECT assignment_id
                                 FROM conf_participant_list
                                 WHERE (participant_id = @particip_id)
                                 AND (conf_reservation_id =
                                @ConferenceDayReservationId))

    IF (@AssignmentId is null)
      BEGIN
        ;THROW 52000, 'Participant has not been assigned to
        appropriate conference day reservation. Cannot assign to him to lecture.',1
      END

    INSERT INTO lecture_participant_list(lecture_reservation_id,participant_id)
    VALUES(@reservation_id,@particip_id)
  END
```

ConfReservation – procedura rejestrująca nową rezerwację na dzień konferencji

```
CREATE PROCEDURE [dbo].[ConfReservation]
  @day_id smallint,
  @client int,
  @places smallint
  AS
    BEGIN
      SET NOCOUNT ON;
      INSERT INTO conf_reservation(conf_day_id,client_id,places_reserved,reservation_date)
      VALUES(@day_id,@client, @places, GETDATE())
    END
```

LectureReservation – procedura rejestrująca nową rezerwację na warsztat

```
   CREATE PROCEDURE [dbo].[LectureReservation]
    @lecture_id int,
    @conf_reservation int,
    @places smallint
    AS
      BEGIN
        SET NOCOUNT ON;
        INSERT INTO lecture_reservation(lecture_id,conf_reservation_id, places_reserved)
        VALUES(@lecture_id,@conf_reservation, @places)
      END
```

CancelLectureReservation – procedura odwołująca rezerwację na warsztat

```
CREATE PROCEDURE [dbo].[CancelLectureReservation]
  @LectureReservationId int
AS
BEGIN
  SET NOCOUNT ON;

  IF((SELECT cancelled FROM lecture_reservation WHERE
lecture_reservation_id=@LectureReservationId)=1)
    BEGIN
      ;THROW 52000, 'This reservation has already been cancelled', 1
    END
  ELSE
      BEGIN TRY
        BEGIN TRAN
            DELETE FROM lecture_participant_list
            WHERE lecture_participant_list.lecture_reservation_id=@LectureReservationId

            UPDATE lecture_reservation
            SET lecture_reservation.cancelled=1
            WHERE lecture_reservation.lecture_reservation_id=@LectureReservationId
        COMMIT TRAN
      END TRY
      BEGIN CATCH
        print error_message()
        ROLLBACK TRANSACTION
      END CATCH
END
```

CancelConfDayReservation – procedura odwołująca rezerwację na dzień konferencji, powoduje także wywołanie procedury odwołującej rezerwacje na warsztaty tego dnia oraz usuwa uczestników z odpowiednich list

```
CREATE PROCEDURE [dbo].[CancelConfDayReservation]
  @ConfReservationId int
AS
BEGIN
  SET NOCOUNT ON;

  IF not exists(SELECT * FROM conf_reservation
              WHERE conf_reservation.conf_reservation_id= @ConfReservationId)
    BEGIN
       DECLARE @message varchar(100) = 'Cannot cancel conference day reservation '
       +@ConfReservationId+ '. It does not exists'
       ;THROW 51000,@message,1
    END

  ELSE
    IF ((SELECT cancelled FROM conf_reservation
          WHERE conf_reservation_id = @ConfReservationId) = 1)
      BEGIN
        ;THROW 52000,'This reservation has already been cancelled.',1
      END
    ELSE
       DECLARE @LectureReservationId int
       DECLARE curs CURSOR LOCAL FOR
                 SELECT lecture_reservation.lecture_reservation_id
                 FROM lecture_reservation
                 WHERE lecture_reservation.conf_reservation_id=
                @ConfReservationId
       OPEN curs
       BEGIN TRY
           BEGIN TRAN

              FETCH NEXT FROM curs INTO @LectureReservationId

              WHILE @@FETCH_STATUS = 0
                BEGIN
                   BEGIN TRY
                      exec CancelLectureReservation @LectureReservationId
                   END TRY
                   BEGIN CATCH
                       PRINT 'Reservation '+@LectureReservationId+' has been
                       removed from database becouse of cancelling reservation'
                   END CATCH
                   FETCH NEXT FROM curs INTO @LectureReservationId
                END

      --cancelling participant'sreservations
             DELETE FROM conf_participant_list
             WHERE conf_reservation_id = @ConfReservationId

             UPDATE conf_reservation
             SET cancelled = 1
             WHERE conf_reservation_id = @ConfReservationId
      CLOSE curs
      DEALLOCATE curs
      COMMIT TRAN
      END TRY
      BEGIN CATCH
          CLOSE curs
          DEALLOCATE curs
          print error_message()
          ROLLBACK TRANSACTION
      END CATCH

END
```

CancelUnpaidConfDayReservation – procedura odwołująca nieopłacone rezerwacje konferencji

```
CREATE PROCEDURE [dbo].[CancelUnpaidConfDayReservation]

AS
BEGIN
  SET NOCOUNT ON;

  DECLARE curs CURSOR LOCAL FOR
    (SELECT cdpi.conf_reservation_id, cdpi.reservation_date
    FROM ConfDayPayments as cdpi
    LEFT OUTER JOIN ToPay as pai
    ON (cdpi.conf_reservation_id=pai.[Conference day reservation id])
    AND (pai.[Paid money] < (cdpi.[Conference day act price]+cdpi.[Lectures act price]))
    INNER JOIN conf_reservation as cr
    ON (cdpi.conf_reservation_id=cr.conf_reservation_id)
    AND (isnull(cr.cancelled,0)=0))

  DECLARE @ReservationID int, @ReservationDate date

  OPEN curs
    FETCH NEXT FROM curs INTO @ReservationID, @ReservationDate
    WHILE @@FETCH_STATUS=0
    BEGIN
      IF(DATEDIFF(day, @ReservationDate, GETDATE())>7)
        BEGIN
           exec CancelConfDayReservation @ReservationID
        END
      FETCH NEXT FROM curs INTO @ReservationID, @ReservationDate
    END
  CLOSE curs
  DEALLOCATE curs
END
```

SetLecturePlacesNumber – procedura zmieniająca liczbę miejsc zarezerwowanych na warsztat

```
CREATE PROCEDURE [dbo].[SetLecturePlacesNumber]
  @LectureReservationId int,
  @PlacesAmount int
AS
BEGIN
  SET NOCOUNT ON;
  UPDATE lecture_reservation
  SET places_reserved = @PlacesAmount
  WHERE lecture_reservation_id = @LectureReservationId
END
```

ReduceConfPlacesNumber – procedura zmniejszająca liczbę miejsc zarezerwowanych na konferencję , w razie próby powiększenia liczby miejsc zaleca się dokonanie nowej rezerwacji na określoną liczbę osób aby uniemożliwić obliczenie ceny w momencie dodania osób wg stawki z czasu pierwotnej rezerwacji

```
CREATE PROCEDURE [dbo].[ReduceConfPlacesNumber]
  @ConfReservationId int,
  @PlacesAmount int
AS
BEGIN
  SET NOCOUNT ON;

  IF((SELECT places_reserved
      FROM conf_reservation
      WHERE conf_reservation_id=@ConfReservationId)<@PlacesAmount)
  BEGIN
    ;THROW 52000, 'You can only make a new reservation to add places.',1
  END
  ELSE
  BEGIN
      UPDATE conf_reservation
      SET places_reserved=@PlacesAmount
      WHERE conf_reservation_id=@ConfReservationId
  END
END
```

DelFromConfParticipantList – procedura usuwająca uczestnika z listy na dany dzień konferencji , powoduje także usunięcie go z listy uczestników warsztatu

```
CREATE PROCEDURE [dbo].[DelFromConfParticipantList]
  @ConfReservationId int,
  @ParticipantID int
AS
BEGIN
  SET NOCOUNT ON;
  BEGIN TRY
    BEGIN TRAN
      DELETE FROM lecture_participant_list
      WHERE participant_id=@ParticipantID
      AND lecture_reservation_id=(SELECT lecture_reservation_id
                                  FROM lecture_reservation
                                  WHERE conf_reservation_id=@ConfReservationId)

      DELETE FROM conf_participant_list
      WHERE participant_id=@ParticipantID
      AND conf_reservation_id=@ConfReservationId

    COMMIT TRAN
  END TRY
  BEGIN CATCH
    print error_message()
    ROLLBACK TRANSACTION
  END CATCH
END
```

DelFromLectureParticipantList – procedura usuwająca uczestnika z listy uczestników warsztatu

```
CREATE PROCEDURE [dbo].[DelFromLectureParticipantList]
  @LectureReservationId int,
  @ParticipantID int
AS
BEGIN
  SET NOCOUNT ON;
  BEGIN TRY
    BEGIN TRAN
      DELETE FROM lecture_participant_list
      WHERE lecture_reservation_id=@LectureReservationId
      AND participant_id=@ParticipantID
    COMMIT TRAN

  END TRY
  BEGIN CATCH
    print error_message()
    ROLLBACK TRANSACTION
  END CATCH
END
```

ExtendPlacesForConfDay – procedura zwiększająca liczbę miejsc dostępnych na dzień konferencji

```
    CREATE PROCEDURE [dbo].[ExtendPlacesForConfDay]
      @ConfDayId int,
      @PlacesAmount int
    AS
    BEGIN
      SET NOCOUNT ON;

      IF(@PlacesAmount<= (SELECT places FROM conf_day WHERE conf_day_id=@ConfDayId))
      BEGIN
        ;THROW 52000, 'You cannot narrow down number of places for conference day', 1
      END
      ELSE
          UPDATE conf_day
          SET places=@PlacesAmount
          WHERE conf_day_id=@ConfDayId
    END
```

ExtendPlacesForLecture – procedura zwiększająca liczbę miejsc dostępnych na warsztat

```
CREATE PROCEDURE [dbo].[ExtendPlacesForLecture]
  @LectureId int,
  @PlacesAmount int
AS
BEGIN
  SET NOCOUNT ON;

  IF(@PlacesAmount<= (SELECT places FROM lecture WHERE lecture_id=@LectureId))
  BEGIN
    ;THROW 52000, 'You cannot narrow down number of places for lecture', 1
  END
  ELSE
      UPDATE lecture
      SET places=@PlacesAmount
      WHERE lecture_id=@LectureId
END
```

GEN_ConfReservation– procedura umożliwiająca dokonanie rezerwacji na dzień konferencji z datą inną niż dzisiejsza – na potrzeby generatora (w oryginalnej procedurze data rezerwacji ustawiana jest za pomocą wywołania GETDATE())

```
CREATE PROCEDURE [dbo].[GEN_ConfReservation]
  @day_id smallint,
  @client int,
  @places smallint,
  @res_date date
  AS
    BEGIN
       SET NOCOUNT ON;
       INSERT INTO conf_reservation(conf_day_id,client_id, places_reserved,
reservation_date)
       VALUES(@day_id,@client, @places, @res_date)
END
```

7. Triggery :

RefuseAssignmentToCancellReservation – trigger blokujący próbę dodania uczestnika do listy na odwołany dzień konferencji

```
CREATE TRIGGER [dbo].[RefuseAssignmentToCancelledReservation]
ON [dbo].[conf_participant_list]
AFTER INSERT, UPDATE
AS
BEGIN
  DECLARE @ConferenceDayReservationId int = (SELECT conf_reservation_id FROM inserted)

  IF ((SELECT cancelled FROM conf_reservation WHERE conf_reservation_id =
@ConferenceDayReservationId) = 1)
     BEGIN
       ;THROW 52000,'This reservation has been cancelled.',1
       ROLLBACK TRANSACTION
     END
 END
```

RefuseToManyConfParticipants – trigger blokujący próbę dodania większej liczby uczestników do listy na dzień konferencji niż liczba przewidziana przez dokonaną przez klienta rezerwację

```
 CREATE TRIGGER [dbo].[RefuseTooManyConfParticipants]
 ON [dbo].[Conf_participant_list]
 AFTER INSERT, UPDATE
 AS
 BEGIN
  DECLARE @ConferenceDayReservationId int = (SELECT conf_reservation_id FROM inserted)
  DECLARE @reserved smallint = (SELECT places_reserved FROM conf_reservation WHERE
conf_reservation_id = @ConferenceDayReservationId)

   IF (@reserved < (SELECT COUNT(assignment_id) FROM conf_participant_list
                    WHERE (conf_reservation_id = @ConferenceDayReservationId)))
      BEGIN
        ;THROW 53000, 'All places has been reserved for this reservation',1
        ROLLBACK TRANSACTION
      END
  END
```

RefuseToManyLectureParticipants – trigger blokujący próbę dodania większej liczby uczestników do listy na warsztat niż liczba przewidziana przez dokonaną przez klienta rezerwację

```
CREATE TRIGGER [dbo].[RefuseTooManyLectureParticipants]
  ON [dbo].[lecture_participant_list]
    AFTER INSERT, UPDATE
AS
BEGIN
  DECLARE @LectureReservationId int = (SELECT lecture_reservation_id FROM inserted)
  DECLARE @reserved tinyint = (SELECT places_reserved
                FROM lecture_reservation
                WHERE lecture_reservation_id = @LectureReservationId)
  IF (@reserved < (SELECT COUNT(assignment_id)
              FROM lecture_participant_list
              WHERE (lecture_reservation_id = @LectureReservationId)))
    BEGIN
      ;THROW 53000, 'All places for this reservation already reserved',1
      ROLLBACK TRANSACTION
    END
END
```

MinPlacesReservedForParticipants – trigger blokujący próbę zmiany liczby rezerwowanych miejsc na dzień konferencji na niższą niż liczba osób już wpisanych na listę uczestników , aby dokonać tej zmiany należy wcześniej usunąć uczestników z odpowiedniej listy

```
CREATE TRIGGER [dbo].[MinPlacesReservedForParticipants]
 ON [dbo].[conf_reservation]
  AFTER UPDATE
   AS
 BEGIN
   DECLARE @ConferenceDayReservationId int = (SELECT conf_reservation_id FROM inserted)
   DECLARE @PlacesWanted smallint = (SELECT places_reserved FROM inserted)
   DECLARE @PlacesSet smallint = (SELECT COUNT(assignment_id) FROM conf_participant_list
                          WHERE conf_reservation_id = @ConferenceDayReservationId)

   IF (@PlacesSet > @PlacesWanted)
     BEGIN
       DECLARE @message varchar(100) = 'Participants assigned
       to this reservation: '+CAST(@PlacesSet as varchar(10))
      ;THROW 52000,@message,1
       ROLLBACK TRANSACTION
      END
  END
```

UniqueParticipantsOnList – trigger blokujący próbę wpisania dwa razy tego samego uczestnika na listę z jednego dnia konferencji

```
CREATE TRIGGER [dbo].[UniqueParticipantsOnList]
  ON [dbo].[conf_participant_list]
  AFTER INSERT, UPDATE
AS
BEGIN
  DECLARE @ParticipantId int = (SELECT participant_id FROM inserted)
  Declare @ConferenceDayReservationId int = (SELECT conf_reservation_id FROM inserted)

  IF(1 <(SELECT COUNT(assignment_id) FROM conf_particpant_list
        WHERE (conf_reservation_id=@ConferenceDayReservationId)
        AND (participant_id=@ParticipantId)))
    BEGIN
      ;THROW 55000, 'This participant has a place for
      this conference day from this client', 1
      ROLLBACK TRANSACTION
    END
END
```

TheSameDayOfConfAndLecture – trigger blokujący próbę podpięcia do rezerwacji dnia konferencji rezerwacji na warsztat odbywający się innego dnia

```
CREATE TRIGGER [dbo].[TheSameDayOfConfAndLecture]
  ON [dbo].[lecture_reservation]
    AFTER INSERT, UPDATE
AS
BEGIN
  SET NOCOUNT ON;
  DECLARE @ConferenceDayId smallint = (SELECT conf_day_id
                      FROM conf_reservation
                      WHERE conf_reservation_id =
                      (SELECT conf_reservation_id
                      FROM inserted))
  DECLARE @LectureConferenceDayId smallint = (SELECT conf_day_id FROM lecture
                      WHERE lecture_id =
                      (SELECT lecture_id FROM inserted))
  IF(@ConferenceDayId <> @LectureConferenceDayId)
  BEGIN
    ;THROW 52000,'This lecture and conference day must be the same day.',1
    ROLLBACK TRANSACTION
  END
END
```

OneLectureReservationForConfDayReservation – trigger blokujący próbę podpięcia do rezerwacji dnia konferencji więcej niż jednej rezerwacji na warsztat

```
CREATE TRIGGER [dbo].[OneLectureReservationForConfDayReservation]
ON [dbo].[lecture_reservation]
   AFTER INSERT, UPDATE
AS
BEGIN
  DECLARE @ConferenceDayReservationId int = (SELECT conf_reservation_id FROM inserted)
  DECLARE @LectureId int = (SELECT lecture_id FROM inserted)

  IF(1<(SELECT count(lecture_reservation_id) FROM lecture_reservation
      WHERE(lecture_id=@LectureId) AND (conf_reservation_id=@ConferenceDayReservationId)))
    BEGIN
      ;THROW 53000, 'Some lecture places already assigned to this conf day reservation', 1
      ROLLBACK TRANSACTION
    END
END
```

OnlyFutureConfSpecyfication – trigger blokujący próbę dodania do bazy konferencji z datą z przeszłości

```
CREATE TRIGGER [dbo].[OnlyFutureConfSpecyfication]
  ON [dbo].[conference]
AFTER INSERT,UPDATE
AS
BEGIN
  SET NOCOUNT ON;
  DECLARE @Date date = (SELECT begin_date FROM inserted)

  IF((DATEDIFF(day,GETDATE(),@Date) <= 0))
    BEGIN
      ;THROW 52000,' Impossible to specify past conferences.',1
      ROLLBACK TRANSACTION
    END
END
```

DayOfConfWithinLength – trigger blokujący próbę dodania do bazy dnia konferencji z numerem dnia większym niż długość konferencji

```
CREATE TRIGGER [dbo].[DayOfConfWithinLength]
ON [dbo].[conf_day]
AFTER INSERT, UPDATE
AS
BEGIN
  DECLARE @DayNumber tinyint = (SELECT day_of_conference FROM inserted)
  DECLARE @ConferenceId smallint = (SELECT conf_id FROM inserted)
  DECLARE @DaysAmount tinyint = (SELECT conference.length FROM conference
                          WHERE conf_id = @ConferenceId)
  IF (@DayNumber not between 1 AND @DaysAmount)
   BEGIN
     DECLARE @message varchar(100) = 'For this conference has been specified only '
                                    +CAST(@DaysAmount as varchar(3))+' days.'
     ;THROW 52000,@message,1
     ROLLBACK TRANSACTION
   END
END
```

CheckForTwoTheSameConferenceDays – trigger blokujący próbę dodania do bazy dnia konferencji z numerem dnia, dla którego taki dzień w bazie już istnieje

```
CREATE TRIGGER [dbo].[CheckForTwoTheSameConferenceDays]
 ON [dbo].[conf_day]
 AFTER INSERT, UPDATE
 AS
 BEGIN
   DECLARE @DayNumber tinyint = (SELECT day_of_conference FROM inserted)
   DECLARE @ConferenceId smallint = (SELECT conf_id FROM inserted)

   IF ((SELECT COUNT(conf_day_id) FROM conf_day
         WHERE (day_of_conference = @DayNumber) AND (conf_id = @ConferenceId) ) > 1)

     BEGIN
       DECLARE @message varchar(100) = 'Day '+CAST(@DayNumber as varchar(3))+
                          ' already exists for this conference'
       ;THROW 52000,@message,1
       ROLLBACK TRANSACTION
     END
 END
```

LectureMinDuration – trigger blokujący próbę dodania do bazy warsztatu krótszego niż 20 min , pilnuje także aby żaden z nich nie miał czasu zakończenia wcześniejszego niż czas rozpoczęcia

```
CREATE TRIGGER [dbo].[LectureMinDuration]
ON [dbo].[lecture]
AFTER INSERT,UPDATE
AS
BEGIN
  SET NOCOUNT ON;
  DECLARE @start time(0) = (SELECT begin_time FROM inserted)
  DECLARE @end time(0) = (SELECT end_time FROM inserted)

  IF((SELECT DATEDIFF(minute,@start,@end))<20)
   BEGIN
    ;THROW 52000,'Lecture has to last at least 20 minutes.',1
    ROLLBACK TRANSACTION
  END
END
```

CloseReservationsForCancelledReservation– trigger blokujący próbę dokonania rezerwacji na warsztat i przypisania jej do odwołanej rezerwacji dnia konferencji

```
CREATE TRIGGER [dbo].[CloseReservationsForCancelledConfReservation]
ON [dbo].[lecture_reservation]
AFTER INSERT, UPDATE
AS
BEGIN
  DECLARE @ConferenceDayReservationId int = (SELECT conf_reservation_id FROM inserted)

  IF ((SELECT cancelled FROM conf_reservation
      WHERE conf_reservation_id = @ConferenceDayReservationId) = 1)
    BEGIN
       ;THROW 52000,'This conference day reservation has been cancelled.',1
       ROLLBACK TRANSACTION
    END
END
```

RefuseSimultaneousLectures – trigger blokujący próbę wpisania uczestnika na dwa warsztaty trwające jednocześnie

```
CREATE TRIGGER [dbo].[RefuseSimultaneousLectures]
  ON [dbo].[lecture_participant_list]
  AFTER INSERT
AS
BEGIN
  DECLARE @LectureId int = (SELECT lecture_id FROM lecture_reservation
                      WHERE lecture_reservation_id =
                      (SELECT lecture_reservation_id FROM inserted))
  DECLARE @BeginTime time(0) = (SELECT begin_time FROM lecture
                               WHERE lecture_id = @LectureId)
  DECLARE @EndTime time(0) = (SELECT end_time FROM lecture
                         WHERE lecture_id = @LectureId)
  DECLARE @ParticipantId int = (SELECT participant_id FROM lecture_participant_list
                             WHERE lecture_reservation_id =
                             (SELECT lecture_reservation_id FROM inserted))
  DECLARE @ConferenceDayId smallint = (SELECT conf_day_id FROM lecture
                                       WHERE lecture_id = @LectureId)
  DECLARE @tempAssignmentId int
  DECLARE @tempLectureId int
  DECLARE @tempBeginTime time(0)
  DECLARE @tempEndTime time(0)


  DECLARE curs CURSOR LOCAL FOR
    (SELECT lpl.assignment_id FROM lecture_participant_list as lpl
      INNER JOIN lecture_reservation as lr
      ON lpl.lecture_reservation_id = lr.lecture_reservation_id
      INNER JOIN lecture as l
      ON lr.lecture_id = l.lecture_id
      WHERE (l.conf_day_id = @ConferenceDayId) AND (lpl.participant_id = @ParticipantId))


  OPEN curs
    FETCH NEXT FROM curs INTO @tempAssignmentId
    WHILE @@FETCH_STATUS = 0
    BEGIN
      IF @tempAssignmentId <> (SELECT assignment_id FROM inserted)
      BEGIN
        SET @tempLectureId = (SELECT lecture_id FROM lecture_reservation
                    WHERE lecture_reservation_id =
                    (SELECT lecture_reservation_id
                     FROM lecture_participant_list
                     WHERE assignment_id =
                     @tempAssignmentId))

        SET @tempBeginTime = (SELECT begin_time FROM lecture
                    WHERE lecture_id = @tempLectureId)

        SET @tempEndTime = (SELECT end_time FROM lecture
                  WHERE lecture_id = @tempLectureId)

        IF (((@tempBeginTime > @BeginTime) AND(@tempBeginTime < @EndTime))
        or((@tempEndTime > @BeginTime)AND(@tempEndTime < @EndTime))
        or((@BeginTime > @tempBeginTime)AND(@BeginTime < @tempEndTime))
                or((@EndTime > @tempBeginTime) AND(@EndTime < @tempEndTime)))

          BEGIN
            DECLARE @message varchar(100) = 'This participant is assigned
              to lecture: ' +cast(@tempLectureId as varchar(20))+
              ' which is at the same time.'
            CLOSE curs
            DEALLOCATE curs
            ;THROW 54000, @message,1
            ROLLBACK TRANSACTION
          END
      END
      FETCH NEXT FROM curs INTO @tempAssignmentId
    END
  CLOSE curs
  DEALLOCATE curs
END
```

LecturePlacesLessThanForConf – trigger blokujący próbę dodania warsztatu z większą liczbą miejsc niż liczba miejsc na dzień konferencji (i tak nikt poza uczestnikami dnia konferencji nie może wziąć udziału w warsztacie)

```
CREATE TRIGGER [dbo].[LecturePlacesLessThanForConf]
ON [dbo].[lecture]
  AFTER INSERT,UPDATE
  AS
  BEGIN
    SET NOCOUNT ON;
    DECLARE @LecturePlaces tinyint = (SELECT places FROM inserted)
    DECLARE @ConferenceDayPlaces smallint = (SELECT C.places FROM inserted as I
INNER JOIN conf_day as C
                                        ON I.conf_day_id = C.conf_day_id)
    IF(@LecturePlaces > @ConferenceDayPlaces)
      BEGIN
      ;THROW 52000,'Impossible to specify lecture with more places than for
conference day.',1
      ROLLBACK TRANSACTION
  END
END
```

OnlyFutureConfSpecyfication – trigger blokujący próbę dodania do bazy konferencji z datą z przeszłości

```
  CREATE TRIGGER [dbo].[CloseReservationsForFullConfDay]
  ON [dbo].[conf_reservation]
  AFTER INSERT, UPDATE
  AS
  BEGIN
     DECLARE @ConferenceDayId smallint = (SELECT conf_day_id FROM inserted)
     IF (dbo.GetConfPlacesLeft(@ConferenceDayId) < 0)
       BEGIN
         DECLARE @placesLeft smallint = dbo.GetConfPlacesLeft(@ConferenceDayId)
                       + (SELECT places_reserved FROM inserted) DECLARE @message
varchar(100) = 'There are only '
                        +CAST(@placesLeft as varchar(10))+' places left for this
conference day.'
         ;THROW 52000,@message,1
         ROLLBACK TRANSACTION
       END
  END
```

CloseReservationsForFullLecture – trigger blokujący próbę dodania rezerwacji na warsztat na większą liczbę osób niż liczba dostępnych miejsc

```
CREATE TRIGGER [dbo].[CloseReservationsForFullLecture]
ON [dbo].[lecture_reservation]
   AFTER INSERT, UPDATE
AS
BEGIN
  DECLARE @LectureId int = (SELECT lecture_id FROM inserted)
  IF(dbo.GetLecturePlacesLeft(@LectureId) < 0)
  BEGIN
    DECLARE @placesLeft tinyint=dbo.GetLecturePlacesLeft(@LectureId)
                +(SELECT places_reserved FROM INSERTED)
    DECLARE @Message varchar(100) = 'There are '
                    +CAST(@placesLeft as varchar(10))
                    +' places left for this lecture.'
    ;THROW 52000, @Message, 1
    ROLLBACK TRANSACTION
  END
END
```

CloseReservationsForNearConf – trigger blokujący próbę dodania rezerwacji na dzień konferencji bliższej niż 2 tygodnie

```
CREATE TRIGGER [dbo].[CloseReservationsForNearConf]
ON [dbo].[conf_reservation]
AFTER INSERT, UPDATE
AS
   BEGIN
     DECLARE @ConferenceDayId smallint = (SELECT conf_day_id FROM inserted)
     DECLARE @Date date = (SELECT reservation_date FROM inserted)
     DECLARE @ConfBegin date = (SELECT begin_date FROM conference
     WHERE conf_id = (SELECT conf_id FROM conf_day WHERE conf_day_id = @ConferenceDayId))

     IF ((DATEADD(DAY,-14,@ConfBegin)) < @Date)
       BEGIN
          ;THROW 53000,'The conference is starting in less than 2 weeks.',1
          ROLLBACK TRANSACTION
       END
    END
```

CloseReservationsForNoPriceConf – trigger blokujący próbę dodania rezerwacji na dzień konferencji, dla której nie jest określony obowiązujący próg cenowy

```
CREATE TRIGGER [dbo].[CloseReservationsForNoPriceConf]
ON [dbo].[conf_reservation]
AFTER INSERT,UPDATE
AS
 BEGIN
   SET NOCOUNT ON;
    DECLARE @Date date = (SELECT reservation_date FROM inserted)
    DECLARE @Conference_day_id smallint = (SELECT conf_day_id FROM inserted)

    IF(dbo.GetPriceStageForDate(@Date,@Conference_day_id) is null)
     BEGIN
       ;THROW 52000,'No price for this conf day.',1
     END
 END
```

PriceToDateBeforeConfBegin – trigger blokujący próbę dodania progu cenowego z datą późniejszą niż dzień rozpoczęcia konferencji

```
CREATE TRIGGER [dbo].[PriceToDateBeforeConfBegin]
ON [dbo].[conf_day_price]
AFTER INSERT, UPDATE
AS
 BEGIN
   DECLARE @Date date = (SELECT to_date FROM inserted)
   DECLARE @ConferenceStartingDay date = (SELECT C.begin_date FROM inserted as I
INNER JOIN conf_day as CD
                                          ON I.conf_day_id = CD.conf_day_id
INNER JOIN conference as C ON CD.conf_id = C.conf_id)

   IF ((SELECT DATEDIFF(day,@Date,@ConferenceStartingDay)) < 0)
     BEGIN
      ;THROW 52000, 'This price ends after conference begining.',1
      ROLLBACK TRANSACTION
     END
  END
```

UniquePriceForDay – trigger blokujący próbę dodania dwóch progów cenowych na jeden okres

```
CREATE TRIGGER [dbo].[UniquePriceForDay]
ON [dbo].[conf_day_price]
AFTER INSERT, UPDATE
AS
BEGIN
 DECLARE @InfoId int = (SELECT price_id FROM inserted)
 DECLARE @Date date = (SELECT to_date FROM inserted)
 DECLARE @ConferenceDayId smallint = (SELECT conf_day_id FROM inserted)

 IF exists(SELECT price_id FROM conf_day_price
          WHERE ((price_id <> @InfoId)AND(to_date = @Date)AND(conf_day_id =
@ConferenceDayId)))
    BEGIN
     ;THROW 52000, 'Impossible to add second price with the same to_date.',1
      ROLLBACK TRANSACTION
    END
END
```

RefuseCancelledReservationPayment – trigger blokujący próbę opłacenia odwołanej rezerwacji

```
CREATE TRIGGER [dbo].[RefuseCancelledReservationPayment]
ON [dbo].[Payment]
AFTER INSERT, UPDATE
AS
  BEGIN
     SET NOCOUNT ON;
     DECLARE @ReservationId int = (SELECT conf_reservation_id FROM inserted)

     IF((SELECT cancelled FROM conf_reservation WHERE conf_reservation_id =
@ReservationId)=1)
       BEGIN
         ;THROW 52000,'This reservation has been cancelled!',1
         ROLLBACK TRANSACTION
       END
  END
```

8. Generator :

Generator danych wypełniających bazę został zaimplementowany w języku C++ , umożliwiając wywoływanie procedur z danymi tak , aby spełniały wymagania związane ze specyfikacją firmy, która organizuje średnio 2 konferencje w miesiącu, a każda z nich trwa zwykle 2-3 dni,w tym średnio w każ dym dniu są 4 warsztaty. Na każdą konferencję średnio rejestruje się 200 osób.