Podstawy Baz Danych

System zarządzania konferencjami

Jan Chyb, Kacper Sala

Spis treści

**Nie znaleziono żadnych pozycji spisu treści.**

1. **Analiza wymagań**
   1. **Konferencje:**

Mogą być jedno lub kilkudniowe. Zawsze odbywała się w ramach całego dnia. Jeżeli są kilkudniowe, to muszą się odbywać w spójnym przedziale dni, w szczególności nigdy nie może zaistnieć sytuacja, że w trakcie trwania jednej konferencji odbędzie się kilkudniowa przerwa na inną konferencję. Istnieje ustalony limit miejsc oddzielnie na każdy dzień konferencji.

* 1. **Klienci:**

Klientami mogą być indywidualne osoby i firmy. Klienci rejestrują

uczestników na poszczególne dni konferencji i na warsztaty.

* 1. **Uczestnicy**:

Uczestnikami konferencji są osoby. Kilka klientów nie może rejestrować tych samych osób na te same dni konferencji. Firma nie musi podawać od razu listy uczestników, musi jednak od razu ustalić ich liczbę na poszczególne dni i na poszczególne warsztaty. Lista wtedy musi zostać uzupełniona o dane uczestników do 2 tygodni przed rozpoczęciem konferencji. Potrzebnymi danymi dla każdego uczestnika są numer PESEL, Imię i numer legitymacji studenckiej, o ile jest studentem.

* 1. **Warsztaty**:

Z konferencją związane są warsztaty. Nie istnieją warsztaty bez przypisanej konferencji. Każde mają określoną datę, godzinę rozpoczęcia, i godzinę zakończenia. Uczestnicy mogą się na nie rejestrować, albo być zarejestrowani przez klienta, o ile są w dzień odbycia się warsztatów zapisani na konferencję. Uczestnik może na raz brać udział w tylko jednych warsztatach, jednak czas rozpoczęcia jego poprzednich może być równy czasowi rozpoczęcia następnych (system nie będzie uwzględniał przerw na dojście). Istnieje ustalony limit miejsc na każdy warsztat, niezależny od limitu na daną konferencję.

* 1. **Opłaty za konferencję**:

Każdy dzień konferencji ma swoją ustaloną cenę bazową na jednego uczestnika. Cena zależy od terminu rezerwacji, im bliżej konferencji tym mniejsza zniżka. Jeżeli rezerwacja zostanie złożona przed dwoma tygodniami przed początkiem konferencji to wynosi ona 10%, przed jednym tygodniem 5%, póżniej należy zapłacić pełną cenę.Istnieje zniżka studencka, wynosi ona 10% i jest przyznawana na podstawie umieszczenia numeru legitymacji studenckiej w systemie, tylko w ramach opłaty za uczestnika posiadającego takową zniżkę. Na uiszczenie opłat klienci mają czas do tygodnia po rezerwacji, albo, jeżeli nastąpi to wcześniej, do rozpoczęcia konferencji. Jeżeli do tego czasu opłata się nie pojawi to rezerwacja jest anulowana.

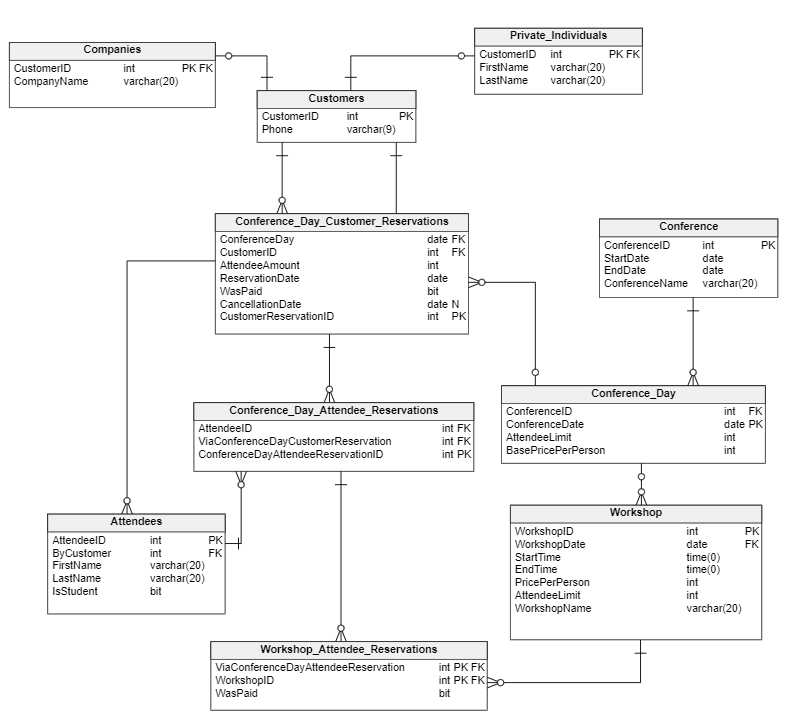
* 1. **Opłaty za warsztaty**:

Cena warsztatów jest stała dla każdego uczestnika. Nie dotyczą jej ani progi, ani zniżka studencka. Opłaty są pobierane bezpośrednio przed warsztatami od uczestników, są jednak archiwizowane w

systemie.

* 1. **Anulowanie rezerwacji**:

W przypadku anulowania rezerwacji na konferencję, zwalniają się miejsca na nią, oraz na warsztaty zajęte przez uczestników zarejestrowanych w ramach anulowanej rezerwacji. Nie można anulować opłaconej rezerwacji.

1. **Schemat bazy danych**
   1. **Diagram bazy danych**
   2. **Tabele**
      1. Customers

Opisuje klientów którzy rezerwują konferencje.

CREATE TABLE Customers  
(  
 CustomerID int NOT NULL IDENTITY (1,1),  
 Phone varchar(9) NOT NULL,  
 CONSTRAINT PhoneCheck CHECK (Phone Like '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'),  
 CONSTRAINT Customers\_pk PRIMARY KEY (CustomerID)  
);

* + 1. Companies

Opisuje firmy które rezerwują konferencje.

CREATE TABLE Companies  
(  
 CustomerID int NOT NULL,  
 CompanyName varchar(20) NOT NULL,  
 CONSTRAINT Companies\_pk PRIMARY KEY (CustomerID)  
);

* + 1. Private\_Individuals

Opisuje osoby prywatne które rezerwują konferencje.

CREATE TABLE Private\_Individuals  
(  
 CustomerID int NOT NULL,  
 FirstName varchar(20) NOT NULL,  
 LastName varchar(20) NOT NULL,  
 CONSTRAINT Private\_Individuals\_pk PRIMARY KEY (CustomerID)  
);

* + 1. Attendees

Opisuje uczestników konferencji. ByCustomer mówi nam przez którego klienta uczestnik został dodany.

CREATE TABLE Attendees  
(  
 AttendeeID int NOT NULL IDENTITY (1,1),  
 ByCustomer int NOT NULL,  
 FirstName varchar(20) NOT NULL,  
 LastName varchar(20) NOT NULL,  
 IsStudent bit NULL,  
 CONSTRAINT Attendees\_pk PRIMARY KEY (AttendeeID)  
);

* + 1. Conference

Opisuje konferencje które organizujemy.

CREATE TABLE Conference  
(  
 ConferenceID int NOT NULL IDENTITY (1,1),  
 StartDate date NOT NULL,  
 EndDate date NOT NULL,  
 ConferenceName varchar(20) NOT NULL,  
 CONSTRAINT ConferenceDateCheck CHECK (StartDate < EndDate),  
 CONSTRAINT Conference\_pk PRIMARY KEY (ConferenceID)  
);

* + 1. Conference\_Day

Opisuje dzień konferencji.

CREATE TABLE Conference\_Day  
(  
 ConferenceID int NOT NULL,  
 ConferenceDate date NOT NULL,  
 AttendeeLimit int NOT NULL,  
 BasePricePerPerson int NOT NULL,  
 CONSTRAINT Conference\_Day\_pk PRIMARY KEY (ConferenceDate)  
);

* + 1. Conference\_Day\_Custumer\_Reservation

Opisuje rezerwacje dnia konferencji przez klienta.

CREATE TABLE Conference\_Day\_Customer\_Reservations  
(  
 ConferenceDay date NOT NULL,  
 CustomerID int NOT NULL,  
 AttendeeAmount int NOT NULL,  
 ReservationDate date NOT NULL,  
 WasPaid bit NOT NULL,   
 CustomerReservationID int NOT NULL IDENTITY (1,1),  
 CONSTRAINT DateCheck CHECK (CancellationDate IS NULL OR ReservationDate < CancellationDate),  
 CONSTRAINT Conference\_Day\_Customer\_Reservations\_pk PRIMARY KEY (CustomerReservationID)  
);

* + 1. Conference\_Day\_Attendee\_Reservation

Opisuje rezerwacje uczestnika konferencji ze względu na dzień. ViaConferenceDayCustomerReservation jest FK który odnosi się do tabeli Conference\_Day\_Custumer\_Reservation.

CREATE TABLE Conference\_Day\_Attendee\_Reservations  
(  
 AttendeeID int NOT NULL,  
 ViaConferenceDayCustomerReservation int NOT NULL,  
 ConferenceDayAttendeeReservationID int NOT NULL IDENTITY (1,1),  
 CONSTRAINT Unique\_Reservation UNIQUE (AttendeeID, ViaConferenceDayCustomerReservation),  
 CONSTRAINT Conference\_Day\_Attendee\_Reservations\_pk PRIMARY KEY (ConferenceDayAttendeeReservationID)  
);

* + 1. Workshop

Opisuje warsztaty które się odbywają w ramach konferencji.

CREATE TABLE Workshop  
(  
 WorkshopID int NOT NULL IDENTITY (1,1),  
 WorkshopDate date NOT NULL,  
 StartTime time(0) NOT NULL,  
 EndTime time(0) NOT NULL,  
 PricePerPerson int NOT NULL,  
 AttendeeLimit int NOT NULL,  
 WorkshopName varchar(20) NOT NULL,  
 CONSTRAINT WorkshopTimeCheck CHECK (StartTime < EndTime),  
 CONSTRAINT Workshop\_pk PRIMARY KEY (WorkshopID)  
);

* + 1. Workshop\_Attendee\_Reservation

Opisuje rezerwacje uczestnika konferencji na poszczególne warsztaty. ViaConferenceDayAttendeeReservation jest to FK który odnosi się do tabeli Conference\_Day\_Attendee\_Reservation.

CREATE TABLE Workshop\_Attendee\_Reservations  
(  
 ViaConferenceDayAttendeeReservation int NOT NULL,  
 WorkshopID int NOT NULL,  
 WasPaid bit NOT NULL,  
 CONSTRAINT Workshop\_Attendee\_Reservations\_pk PRIMARY KEY (ViaConferenceDayAttendeeReservation, WorkshopID)  
);

* 1. **Definicje kluczy obcych**

1. -- foreign keys  
   -- Reference: Attendees\_Customers (table: Attendees)  
   ALTER TABLE Attendees  
    ADD CONSTRAINT Attendees\_Customers  
    FOREIGN KEY (ByCustomer)  
    REFERENCES Customers (CustomerID);  
     
   -- Reference: CDAR\_Attendees (table: Conference\_Day\_Attendee\_Reservations)  
   ALTER TABLE Conference\_Day\_Attendee\_Reservations  
    ADD CONSTRAINT CDAR\_Attendees  
    FOREIGN KEY (AttendeeID)  
    REFERENCES Attendees (AttendeeID);  
     
   -- Reference: CDAR\_CDR (table: Conference\_Day\_Attendee\_Reservations)  
   ALTER TABLE Conference\_Day\_Attendee\_Reservations  
    ADD CONSTRAINT CDAR\_CDR  
    FOREIGN KEY (ViaConferenceDayCustomerReservation)  
    REFERENCES Conference\_Day\_Customer\_Reservations (CustomerReservationID)  
    ON DELETE CASCADE;  
     
   -- Reference: CDAR\_WAR (table: Workshop\_Attendee\_Reservations)  
   ALTER TABLE Workshop\_Attendee\_Reservations  
    ADD CONSTRAINT CDAR\_WAR  
    FOREIGN KEY (ViaConferenceDayAttendeeReservation)  
    REFERENCES Conference\_Day\_Attendee\_Reservations (ConferenceDayAttendeeReservationID)  
    ON DELETE CASCADE;  
     
   -- Reference: CDR\_Customers (table: Conference\_Day\_Customer\_Reservations)  
   ALTER TABLE Conference\_Day\_Customer\_Reservations  
    ADD CONSTRAINT CDR\_Customers  
    FOREIGN KEY (CustomerID)  
    REFERENCES Customers (CustomerID);  
     
   -- Reference: CD\_CDR (table: Conference\_Day\_Customer\_Reservations)  
   ALTER TABLE Conference\_Day\_Customer\_Reservations  
    ADD CONSTRAINT CD\_CDR  
    FOREIGN KEY (ConferenceDay)  
    REFERENCES Conference\_Day (ConferenceDate);  
     
   -- Reference: CD\_Conference (table: Conference\_Day)  
   ALTER TABLE Conference\_Day  
    ADD CONSTRAINT CD\_Conference  
    FOREIGN KEY (ConferenceID)  
    REFERENCES Conference (ConferenceID);  
     
   -- Reference: Company\_Customers (table: Companies)  
   ALTER TABLE Companies  
    ADD CONSTRAINT Company\_Customers  
    FOREIGN KEY (CustomerID)  
    REFERENCES Customers (CustomerID);  
     
   -- Reference: PI\_Customers (table: Private\_Individuals)  
   ALTER TABLE Private\_Individuals  
    ADD CONSTRAINT PI\_Customers  
    FOREIGN KEY (CustomerID)  
    REFERENCES Customers (CustomerID);  
     
   -- Reference: WAR\_Workshop (table: Workshop\_Attendee\_Reservations)  
   ALTER TABLE Workshop\_Attendee\_Reservations  
    ADD CONSTRAINT WAR\_Workshop  
    FOREIGN KEY (WorkshopID)  
    REFERENCES Workshop (WorkshopID);  
     
   -- Reference: Workshop\_CD (table: Workshop)  
   ALTER TABLE Workshop  
    ADD CONSTRAINT Workshop\_CD  
    FOREIGN KEY (WorkshopDate)  
    REFERENCES Conference\_Day (ConferenceDate);
2. **Widoki**
   1. **Customer\_Reservation\_Discount**

CREATE OR ALTER VIEW Customer\_Reservation\_Discount AS  
(  
SELECT CustomerReservationID,  
 *DATEDIFF*(day, ReservationDate, ConferenceDay) as diff,  
 CASE  
 WHEN *DATEDIFF*(day, ReservationDate, ConferenceDay) >= 14 THEN 0.1  
 WHEN *DATEDIFF*(day, Reservationdate, ConferenceDay) >= 7 THEN 0.05  
 ELSE 0  
 END  
 AS Discount  
FROM Conference\_Day\_Customer\_Reservations cdcr  
 )  
GO

* 1. **Attendee\_Reservation\_Value**

CREATE OR ALTER VIEW Attendee\_Reservation\_Value AS  
(  
SELECT ViaConferenceDayCustomerReservation,  
 cdar.AttendeeID,  
 IsStudent,  
 (1 - Discount) \*  
 (0.9 \* *CAST*(BasePricePerPerson AS FLOAT) \* IsStudent + BasePricePerPerson \* (1 - IsStudent)) AS ReservationValue  
FROM Conference\_Day\_Attendee\_Reservations cdar  
 INNER JOIN Attendees a ON cdar.AttendeeID = a.AttendeeID  
 INNER JOIN Conference\_Day\_Customer\_Reservations cdcr  
 ON cdcr.CustomerReservationID = cdar.ViaConferenceDayCustomerReservation  
 INNER JOIN Conference\_Day cd ON cd.ConferenceDate = cdcr.ConferenceDay  
 Inner JOIN Customer\_Reservation\_Discount crd ON crd.CustomerReservationID = cdcr.CustomerReservationID  
 )  
GO

* 1. **Conference\_Payments**

CREATE OR ALTER VIEW Conference\_Payments AS  
(  
SELECT CustomerID, ConferenceDay, *SUM*(ReservationValue) AS ResrvationValue  
FROM Conference\_Day\_Customer\_Reservations cdcr  
 INNER JOIN Attendee\_Reservation\_Value crv  
 ON cdcr.CustomerReservationID = crv.ViaConferenceDayCustomerReservation  
WHERE WasPaid = 1  
GROUP BY CustomerID, ConferenceDay  
 )  
GO

1. **Funkcje**
   1. **conference\_free\_places**

Zwraca liczbę wolnych miejsc na konferencji.

CREATE OR ALTER FUNCTION *func\_conference\_free\_places*(@ConferenceDay date)  
 RETURNS int  
AS  
BEGIN  
 DECLARE @AttendeeLimit int  
 SELECT @AttendeeLimit = AttendeeLimit  
 FROM Conference\_Day  
 WHERE ConferenceDate = @ConferenceDay  
  
 DECLARE @AttendeeAmount int  
 SELECT @AttendeeAmount = *SUM*(AttendeeAmount)  
 FROM Conference\_Day\_Customer\_Reservations  
 WHERE ConferenceDay = @ConferenceDay  
  
 RETURN *ISNULL*(@AttendeeLimit, 0) - *ISNULL*(@AttendeeAmount, 0)  
END  
GO

* 1. **workshop\_free\_places**

Zwraca liczbę wolnych miejsc na warsztacie.

CREATE OR ALTER FUNCTION *func\_workshop\_free\_places*(@WorkshopDay date)  
 RETURNS int  
AS  
BEGIN  
 DECLARE @WorkshopID int  
 DECLARE @AttendeeLimit int  
 SELECT @AttendeeLimit = AttendeeLimit, @WorkshopID = WorkshopID  
 FROM Workshop  
 WHERE WorkshopDate = @WorkshopDay  
  
 DECLARE @AttendeeAmount int  
 SELECT @AttendeeAmount = *COUNT*(WorkshopID)  
 FROM Workshop\_Attendee\_Reservations  
 WHERE WorkshopID = @WorkshopID  
  
 RETURN *ISNULL*(@AttendeeLimit, 0) - *ISNULL*(@AttendeeAmount, 0)  
END  
GO

* 1. **workshop\_list\_for\_attendee**

Zwraca tabele wszystkich warsztatów na które jest zapisany uczestnik.

CREATE OR ALTER FUNCTION *func\_workshop\_list\_for\_attendee*(@AttendeeID int)  
 RETURNS @ReturnTable TABLE  
 (  
 WorkshopID int,  
 WorkshopDate date,  
 StartTime time(0),  
 EndTime time(0),  
 PricePerPerson int,  
 AttendeeLimit int,  
 WorkshopName varchar(20)  
 )  
AS  
BEGIN  
 INSERT INTO @ReturnTable  
 SELECT \*  
 FROM Workshop AS w  
 WHERE w.WorkshopID IN (SELECT war.WorkshopID  
 FROM Workshop\_Attendee\_Reservations AS war  
 JOIN Conference\_Day\_Attendee\_Reservations AS cdar  
 ON war.ViaConferenceDayAttendeeReservation =  
 cdar.ConferenceDayAttendeeReservationID  
 WHERE cdar.AttendeeID = @AttENDEEID)  
 RETURN;  
END  
GO

* 1. **conference\_day\_attendees**

Zwraca tabele ID uczestników dnia konferencji.

CREATE OR ALTER FUNCTION *func\_conference\_day\_attendees*(@ConferenceDate date)  
 RETURNS @ReturnTable TABLE  
 (  
 AttendeeID int  
 )  
AS  
BEGIN  
 INSERT INTO @ReturnTable  
 SELECT AttendeeID  
 FROM Conference\_Day\_Attendee\_Reservations AS cdar  
 JOIN Conference\_Day\_Customer\_Reservations AS cdcr  
 ON cdar.ViaConferenceDayCustomerReservation = cdcr.CustomerReservationID  
 WHERE cdcr.ConferenceDay = @ConferenceDate  
 RETURN;  
END  
GO

* 1. **conference\_days**

Zwraca tabele wszystkich dni które są zarezerwowane przez jedną konferencję.

CREATE OR ALTER FUNCTION *func\_conference\_days*(@ConferenceID int)  
 RETURNS @ReturnTable TABLE  
 (  
 ConferenceID int,  
 ConferenceDate date,  
 AttendeeLimit int,  
 BasePricePerPerson int  
 )  
AS  
BEGIN  
 INSERT INTO @ReturnTable  
 SELECT \*  
 FROM Conference\_Day AS cd  
 WHERE @ConferenceID = cd.ConferenceID  
 RETURN;  
END  
GO

* 1. **workshop\_list\_by\_day**

Zwraca listę wszystkich warsztatów które odbywają się w danym dniu.

CREATE OR ALTER FUNCTION *func\_workshop\_list\_by\_day*(@Day date)  
 RETURNS @ReturnTable TABLE  
 (  
 WorkshopID int  
 )  
AS  
BEGIN  
 INSERT INTO @ReturnTable  
 SELECT WorkshopID  
 FROM Workshop  
 WHERE WorkshopDate = @Day  
 RETURN  
END  
GO

1. **Procedury**
   1. **proc\_new\_conference**

Tworzy nową konferencję

CREATE OR ALTER PROCEDURE *proc\_new\_conference*(@ConferenceName varchar(20),  
 @StartDate date,  
 @EndDate date)  
AS  
BEGIN  
 SET NOCOUNT ON;  
 INSERT INTO Conference(ConferenceName, StartDate, EndDate) VALUES (@ConferenceName, @StartDate, @EndDate)  
END  
GO

* 1. **proc\_new\_conference\_day**

Tworzy nowy rekord dnia konferencji.

CREATE OR ALTER PROCEDURE *proc\_new\_conference\_day*(@ConferenceID int,  
 @ConferenceDate date,  
 @AttendeeLimit int,  
 @BasePricePerPerson int)  
AS  
BEGIN  
 SET NOCOUNT ON;  
 INSERT INTO Conference\_Day(ConferenceID, ConferenceDate, AttendeeLimit, BasePricePerPerson)  
 VALUES (@ConferenceID, @ConferenceDate, @AttendeeLimit, @BasePricePerPerson)  
END  
GO

* 1. **proc\_new\_company**

Tworzy rekord nowej firmy która jest klientem.

CREATE OR ALTER PROCEDURE *proc\_new\_company*(@CompanyName varchar(20),  
 @Phone varchar(9))  
AS  
BEGIN  
 SET NOCOUNT ON;  
 INSERT INTO Customers(Phone) VALUES (@Phone)  
 DECLARE @CustomerID INT  
 SET @CustomerID = *@@IDENTITY* INSERT INTO Companies(CustomerID, CompanyName) VALUES (@CustomerID, @CompanyName)  
END  
GO

* 1. **proc\_new\_private\_individual**

Tworzy nową osobę prywatną która jest klientem.

CREATE OR ALTER PROCEDURE *proc\_new\_private\_individual*(@FirstName varchar(20),  
 @LastName varchar(20),  
 @Phone varchar(9))  
AS  
BEGIN  
 SET NOCOUNT ON;  
 INSERT INTO Customers(Phone) VALUES (@Phone)  
 DECLARE @CustomerID INT  
 SET @CustomerID = *@@IDENTITY* INSERT INTO Private\_Individuals(CustomerID, FirstName, LastName) VALUES (@CustomerID, @FirstName, @LastName)  
END  
GO

* 1. **proc\_new\_attendee**

Tworzy nowego potencjalnego uczestnika konferencji.

CREATE OR ALTER PROCEDURE *proc\_new\_attendee*(@ByCustomer integer,  
 @FirstName varchar(20),  
 @LastName varchar(20),  
 @IsStudent bit)  
AS  
BEGIN  
 SET NOCOUNT ON;  
 INSERT INTO Attendees(ByCustomer, FirstName, LastName, IsStudent)  
 VALUES (@ByCustomer, @FirstName, @LastName, @IsStudent)  
END  
GO

* 1. **proc\_new\_workshop**

Tworzy nowy warsztat.

CREATE OR ALTER PROCEDURE *proc\_new\_workshop*(@WorkshopDate date,  
 @StartTime time(0),  
 @EndTime time(0),  
 @PricePerPerson int,  
 @AttendeeLimit int,  
 @WorkshopName varchar(20))  
AS  
BEGIN  
 SET NOCOUNT ON;  
 INSERT INTO Workshop(WorkshopDate, StartTime, EndTime, PricePerPerson, AttendeeLimit, WorkshopName)  
 VALUES (@WorkshopDate, @StartTime, @EndTime, @PricePerPerson, @AttendeeLimit, @WorkshopName)  
END  
GO

* 1. **proc\_new\_customer\_conference\_day\_reservation**

Tworzy nowy rekord dotyczący rezerwacji dnia konferencji przez klienta.

CREATE OR ALTER PROCEDURE *proc\_new\_customer\_conference\_day\_reservation*(@ConferenceDay date,  
 @CustomerID int,  
 @AttendeeAmount int,  
 @ReservationDate date,  
 @WasPaid bit)  
AS  
BEGIN  
 SET NOCOUNT ON;  
 INSERT INTO Conference\_Day\_Customer\_Reservations(ConferenceDay, CustomerID, AttendeeAmount, ReservationDate, WasPaid)  
 VALUES (@ConferenceDay, @CustomerID, @AttendeeAmount, @ReservationDate, @WasPaid)  
END  
GO

* 1. **proc\_cancel\_customer\_conference\_day\_reservation**

Usuwa rezerwację dnia konferencji.

CREATE OR ALTER PROCEDURE *proc\_cancel\_customer\_conference\_day\_reservation*(@CustomerReservationID int)  
AS  
BEGIN  
 SET NOCOUNT ON;  
 DELETE  
 FROM Workshop\_Attendee\_Reservations  
 WHERE ViaConferenceDayAttendeeReservation IN  
 (SELECT ConferenceDayAttendeeReservationID  
 FROM Conference\_Day\_Attendee\_Reservations as tmp  
 WHERE tmp.ViaConferenceDayCustomerReservation = @CustomerReservationID)  
 DELETE FROM Conference\_Day\_Attendee\_Reservations WHERE ViaConferenceDayCustomerReservation = @CustomerReservationID  
 DELETE FROM Conference\_Day\_Customer\_Reservations WHERE CustomerReservationID = @CustomerReservationID  
END  
GO

* 1. **proc\_new\_attendee\_conference\_day\_reservation**

Tworzy nową rezerwację miejsca na odpowiedni dzień konferencji dla uczestnika.

CREATE OR ALTER PROCEDURE *proc\_new\_attendee\_conference\_day\_reservation*(@AttendeeID int,  
 @ViaConferenceDayCustomerReservation int)  
AS  
BEGIN  
 SET NOCOUNT ON;  
 INSERT INTO Conference\_Day\_Attendee\_Reservations(AttendeeID, ViaConferenceDayCustomerReservation)  
 VALUES (@AttendeeID, @ViaConferenceDayCustomerReservation)  
END  
GO

* 1. **proc\_cancel\_attendee\_conference\_day\_reservation**

Usuwa rezerwację miejsca na odpowiedni dzień konferencji dla uczestnika.

CREATE OR ALTER PROCEDURE *proc\_cancel\_attendee\_conference\_day\_reservation*(@ConferenceDayAttendeeReservationID int)  
AS  
BEGIN  
 SET NOCOUNT ON;  
 DELETE  
 FROM Conference\_Day\_Attendee\_Reservations  
 WHERE ConferenceDayAttendeeReservationID = @ConferenceDayAttendeeReservationID  
END  
GO

* 1. **proc\_new\_workshop\_attendee\_reservation**

Tworzy nową rezerwację miejsca na odpowiedni warsztat dla uczestnika konferencji.

CREATE OR ALTER PROCEDURE *proc\_new\_workshop\_attendee\_reservation*(@ViaConferenceDayAttendeeReservation int,  
 @WorkshopID int,  
 @WasPaid bit)  
AS  
BEGIN  
 SET NOCOUNT ON;  
 INSERT INTO Workshop\_Attendee\_Reservations(ViaConferenceDayAttendeeReservation, WorkshopID, WasPaid)  
 VALUES (@ViaConferenceDayAttendeeReservation, @WorkshopID, @WasPaid)  
END  
GO

* 1. **proc\_cancel\_workshop\_attendee\_reservation**

Usuwa rezerwację miejsca na warsztat.

CREATE OR ALTER PROCEDURE *proc\_cancel\_workshop\_attendee\_reservation*(@ViaConferenceDayAttendeeReservation int)  
AS  
BEGIN  
 SET NOCOUNT ON;  
 DELETE  
 FROM Workshop\_Attendee\_Reservations  
 WHERE ViaConferenceDayAttendeeReservation = @ViaConferenceDayAttendeeReservation  
END  
GO

* 1. **proc\_add\_conference\_day\_payment**

Aktualizuje stan dnia konferencji na zapłacony.

CREATE OR ALTER PROCEDURE *proc\_add\_conference\_day\_payment*(@ConferenceDay date)  
AS  
BEGIN  
 SET NOCOUNT ON;  
 UPDATE Conference\_Day\_Customer\_Reservations  
 SET WasPaid = 1  
 WHERE ConferenceDay = @ConferenceDay  
END  
GO

* 1. **proc\_add\_workshop\_payment**

Aktualizuje stan warsztatu na zapłacony.

CREATE OR ALTER PROCEDURE *proc\_add\_workshop\_payment*(@ConferenceDayAttendeeReservationID int)  
AS  
BEGIN  
 SET NOCOUNT ON;  
 UPDATE Workshop\_Attendee\_Reservations  
 SET WasPaid = 1  
 WHERE ViaConferenceDayAttendeeReservation = @ConferenceDayAttendeeReservationID  
END  
GO

1. **Triggery**
   1. **trig\_conference\_day\_within\_conference**

Sprawdza czy dzień konferencji jest we wcześniej zdefiniowanym czasie całej konferencji.

CREATE OR ALTER TRIGGER trig\_conference\_day\_within\_conference  
 ON Conference\_Day  
 AFTER INSERT  
 AS  
BEGIN  
 SET NOCOUNT ON;  
 IF *EXISTS*(  
 SELECT \*  
 FROM inserted AS i  
 JOIN Conference AS c ON c.ConferenceID = i.ConferenceID  
 WHERE i.ConferenceDate < c.StartDate  
 OR i.ConferenceDate > c.EndDate  
 )  
 BEGIN  
 THROW 51000, 'Conference day(s) is(are) not within Conference duration.', 1  
 END  
END  
GO

* 1. **trig\_not\_enough\_places\_conference**

Sprawdza czy jest wystarczająco dużo miejsc na zarezerwowanie konferencji.

CREATE OR ALTER TRIGGER trig\_not\_enough\_places\_conference  
 ON Conference\_Day\_Customer\_Reservations  
 AFTER INSERT  
 AS  
BEGIN  
 SET NOCOUNT ON;  
 IF *EXISTS*(  
 SELECT \*  
 FROM inserted AS i  
 WHERE (dbo.*func\_conference\_free\_places*(i.ConferenceDay) < 0)  
 )  
 BEGIN  
 THROW 51000, 'Not enough places for conference', 1;  
 END  
END  
GO

* 1. **trig\_not\_enough\_places\_workshop**

Sprawdza czy jest wystarczająco dużo miejsca aby zarezerwować miejsce na warsztat.

CREATE OR ALTER TRIGGER trig\_not\_enough\_places\_workshop  
 ON Workshop\_Attendee\_Reservations  
 AFTER INSERT  
 AS  
BEGIN  
 SET NOCOUNT ON;  
 IF *EXISTS*(SELECT \*  
 FROM inserted AS i  
 JOIN Workshop AS w  
 ON i.WorkshopID = w.WorkshopID  
 WHERE (dbo.*func\_workshop\_free\_places*(w.WorkshopDate) < 0))  
 BEGIN  
 THROW 51000, 'Not enough places for workshop', 1;  
 END  
END  
GO

* 1. **trig\_workshop\_reservation\_corresponds\_with\_conference**

Sprawdza czy uczestnik chcący zarejestrować się na warsztat jest w tym samym dniu uczestnikiem konferencji.

CREATE OR ALTER TRIGGER trig\_workshop\_reservation\_corresponds\_with\_conference  
 ON Workshop\_Attendee\_Reservations  
 AFTER INSERT  
 AS  
BEGIN  
 SET NOCOUNT ON;  
 IF NOT *EXISTS*(  
 SELECT \*  
 FROM inserted AS i  
 JOIN Conference\_Day\_Attendee\_Reservations as cdar  
 ON i.ViaConferenceDayAttendeeReservation = cdar.ConferenceDayAttendeeReservationID  
 JOIN Conference\_Day\_Customer\_Reservations AS cdcr  
 ON cdcr.CustomerReservationID = cdar.ViaConferenceDayCustomerReservation  
 JOIN Workshop as w  
 ON i.WorkshopID = w.WorkshopID  
 WHERE w.WorkshopDate = cdcr.ConferenceDay  
 )  
 BEGIN  
 THROW 51000, 'Can not reserve workshop without attending the conference on the same day', 1;  
 END  
END  
GO

* 1. **trig\_attendee\_mentioned\_by\_customer**

Sprawdza czy uczestnik chcący wpisać się na konferencję jest faktycznie podanym uczestnikiem przez klienta.

CREATE OR ALTER TRIGGER trig\_attendee\_mentioned\_by\_customer  
 ON Conference\_Day\_Attendee\_Reservations  
 AFTER INSERT  
 AS  
BEGIN  
 SET NOCOUNT ON;  
 IF NOT *EXISTS*(SELECT \*  
 FROM inserted AS i  
 JOIN Attendees AS a ON i.AttendeeID = a.AttendeeID  
 JOIN Conference\_Day\_Customer\_Reservations AS cdcr  
 ON i.ViaConferenceDayCustomerReservation = cdcr.CustomerReservationID  
 Where cdcr.CustomerID = a.ByCustomer)  
 BEGIN  
 THROW 51000, 'Tried to attend conference but is not mentioned by the customer reserving conference.', 1  
 END  
END  
GO

* 1. **trig\_overlapping\_workshops**

Sprawdza czy uczestnik chce zarezerwować warsztat który nachodzi w czasie na inny na który już jest zapisany.

CREATE OR ALTER TRIGGER trig\_overlapping\_workshops  
 ON Workshop\_Attendee\_Reservations  
 AFTER INSERT  
 AS  
BEGIN  
 SET NOCOUNT ON;  
 DECLARE @AttendeeID int  
 SELECT @AttendeeID = cdar.AttendeeID  
 FROM Conference\_Day\_Attendee\_Reservations AS cdar  
 JOIN Workshop\_Attendee\_Reservations AS war  
 ON ConferenceDayAttendeeReservationID = war.ViaConferenceDayAttendeeReservation  
 IF *EXISTS*(SELECT \*  
 FROM inserted AS i  
 JOIN Workshop AS w ON i.WorkshopID = w.WorkshopID  
 JOIN dbo.*func\_workshop\_list\_for\_attendee*(@AttendeeID) AS awl ON w.WorkshopDate = awl.WorkshopDate  
 WHERE awl.WorkshopID <> w.WorkshopID  
 AND (w.StartTime < awl.StartTime AND awl.StartTime < w.EndTime OR  
 w.StartTime < awl.EndTime AND awl.EndTime < w.EndTime)  
 )  
 BEGIN  
 THROW 51000, 'Workshop overlapping with another workshop.', 1;  
 END  
END  
GO

* 1. **trig\_cancel\_payed\_conference**

Sprawdza czy nie próbujemy usunąć już opłaconej konferencji.

CREATE OR ALTER TRIGGER trig\_cancel\_payed\_conference  
 ON Conference\_Day\_Customer\_Reservations  
 AFTER DELETE  
 AS  
BEGIN  
 SET NOCOUNT ON;  
 IF *EXISTS*(SELECT \*  
 FROM deleted AS d  
 WHERE d.WasPaid = 1)  
 BEGIN  
 THROW 51000, 'Trying to cancel already paid reservation.', 1;  
 END  
END  
GO

1. **Indeksy**
2. **Generator danych**