

Podstawy baz danych

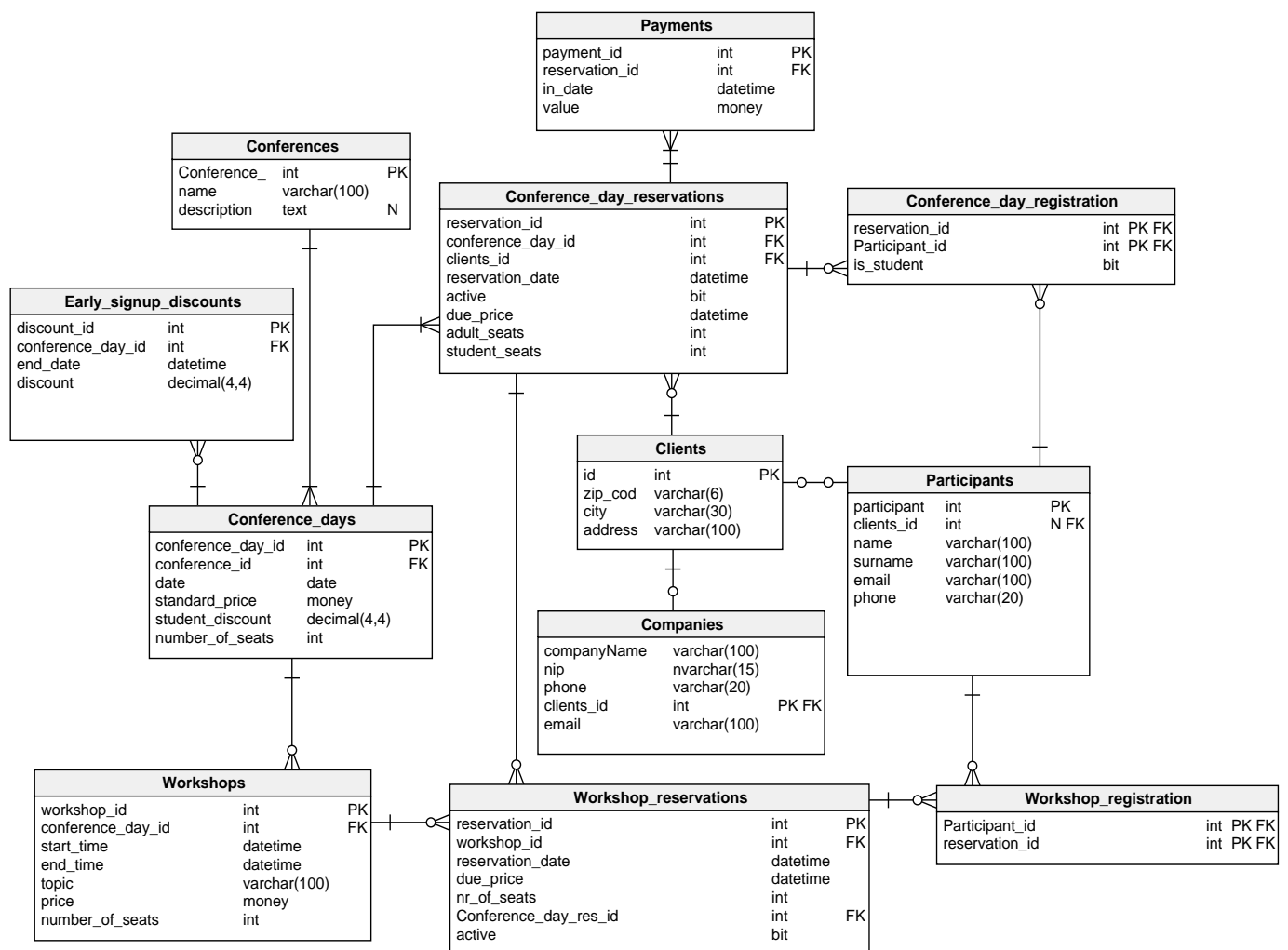
Projekt konferencje

Agnieszka Dutka, Maciek Trątnowiecki

AGH, Styczeń 2020

Objaśnienie schematu bazy

- Clients - Reprezentuje klientów chcących opłacić miejsca na konferencjach i warsztatach. Klientem może być zarówno firma, jak i osoba prywatna. W zależności od tego dane klienta reprezentowane są przez odpowiednią relację w bazie.
- Companies - Jeśli klient jest firmą, przechowuje jego dane.
- Participants - Jeśli klient jest osobą prywatną, przechowuje jego dane.
- Conferences - Reprezentuje konferencję z którą powiązane są odpowiednie dni konferencyjne, oraz warsztaty.
- Conference_days - Reprezentuje pojedynczy dzień konferencji. Powiązana jest z nim ustalona opłata za uczestnictwo. Zniżki obowiązujące w zależności od daty rejestracji zawarte są w relacji Early_Signup_Discounts.
- Early_Signup_Discounts - Odpowiada za informację o tabeli zniżek na dany dzień konferencyjny. Pojedyncza zniżka przechowywana jest w krotce z atrybutami w postaci procentowej obniżki ceny standardowej, oraz ostatniego dnia w którym obowiązuje.
- Conference_day_reservations - Realizuje rezerwacje na poszczególny dzień konferencji. Każda rezerwacja powiązana jest z klientem, który ją opłaca. Za powiązanie rezerwacji z uczestnikiem odpowiada osobna relacja. Zawiera także pole due_price określające termin płatności. Atrybut active odpowiada za możliwość rezygnacji z podjętej rezerwacji (uznaliśmy, że usuwanie krotki z bazy może nie być optymalnym rozwiązaniem, jako że zawarte w niej dane mogą jeszcze być przydatne z punktu widzenia logiki biznesowej). Atrybuty adult_seats i student_seats służą do liczenia kosztu podjęcia rezerwacji przed powiązaniem jej z uczestnikami konferencji.
- Conference_day_registration - Wiąże rezerwację z uczestnikami konferencji. Atrybut is_student informuje, czy danemu uczestnikowi przysługuje zniżka studencka.
- Payments - Przechowuje informacje o wpływach pieniężnych powiązanych z daną rejestracją.
- Workshops - Reprezentuje warsztaty odbywające się w trakcie odpowiednich dni konferencyjnych.
- Workshops_reservations - Opisuje rezerwacje na warsztaty w sposób analogiczny do rezerwacji na konferencje.
- Workshops_registrations - Łączy rezerwację z uczestnikami w sposób analogiczny do dni konferencyjnych.



Implementacja

```
1000 -- tables
1001 -- Table: Clients
1002 CREATE TABLE Clients (
1003     id int NOT NULL IDENTITY,
1004     zip_code varchar(6) NOT NULL,
1005     city varchar(30) NOT NULL,
1006     address varchar(100) NOT NULL,
1007     CONSTRAINT Clients_pk PRIMARY KEY (id)
1008 );
1009
1010 -- Table: Companies
1011 CREATE TABLE Companies (
1012     companyName varchar(100) NOT NULL,
1013     nip nvarchar(15) NOT NULL CHECK ((nip not like '%[0-9]%' ) and (LEN(nip) = 10) and (nip
1014     not like '0%' or nip like '1%')),
1015     phone varchar(20) NOT NULL,
1016     clients_id int NOT NULL,
1017     email varchar(100) NOT NULL CHECK (email like '%_@_%._%'),
1018     CONSTRAINT unique_nip UNIQUE (nip),
1019     CONSTRAINT checkNip CHECK (dbo.IsValidNip(nip) = 1),
1020     CONSTRAINT Companies_pk PRIMARY KEY (clients_id)
1021 );
1022
1023 CREATE INDEX companies_client_id on Companies (clients_id ASC)
1024 ;
1025
1026 CREATE INDEX companies_nip on Companies (nip ASC)
1027 ;
1028
1029 -- Table: Conference_day_registration
1030 CREATE TABLE Conference_day_registration (
1031     reservation_id int NOT NULL,
1032     Participant_id int NOT NULL,
1033     is_student bit NOT NULL DEFAULT 0,
1034     CONSTRAINT Conference_day_registration_pk PRIMARY KEY (reservation_id ,Participant_id)
1035 );
1036
1037 CREATE INDEX day_reg_part_id on Conference_day_registration (Participant_id ASC)
1038 ;
1039
1040 CREATE INDEX day_reg_res_id on Conference_day_registration (reservation_id ASC)
1041 ;
1042
1043 -- Table: Conference_day_reservations
1044 CREATE TABLE Conference_day_reservations (
1045     reservation_id int NOT NULL IDENTITY,
1046     conference_day_id int NOT NULL,
1047     clients_id int NOT NULL,
1048     reservation_date datetime NOT NULL DEFAULT GETDATE() ,
1049     active bit NOT NULL DEFAULT 1,
1050     due_price datetime NOT NULL DEFAULT DATEADD(week, 2, GETDATE()) CHECK (due_price >=
1051     GETDATE()),
1052     adult_seats int NOT NULL DEFAULT 0 CHECK (adult_seats >= 0),
1053     student_seats int NOT NULL DEFAULT 0 CHECK (student_seats >= 0),
1054     CONSTRAINT Conference_day_reservations_pk PRIMARY KEY (reservation_id)
1055 );
1056
1057 CREATE INDEX day_res_day_id on Conference_day_reservations (conference_day_id ASC)
1058 ;
1059
1060 CREATE INDEX day_res_client_id on Conference_day_reservations (clients_id ASC)
1061 ;
1062
1063 CREATE INDEX day_res_pay_deadline on Conference_day_reservations (due_price ASC)
1064 ;
1065
1066 -- Table: Conference_days
1067 CREATE TABLE Conference_days (
1068     conference_day_id int NOT NULL IDENTITY,
1069     conference_id int NOT NULL,
1070     date date NOT NULL DEFAULT GETDATE() ,
1071     standard_price money NOT NULL DEFAULT 0 CHECK (standard_price >= 0),
1072     student_discount decimal(4,4) NOT NULL DEFAULT 0 CHECK (student_discount >= 0),
1073     number_of_seats int NOT NULL DEFAULT 0 CHECK (number_of_seats >= 0),
```

```

1072     CONSTRAINT Conference_days_pk PRIMARY KEY (conference_day_id)
1073 );
1074
1075 CREATE INDEX day_conf_id on Conference_days (conference_id ASC)
1076 ;
1077
1078 — Table: Conferences
1079 CREATE TABLE Conferences (
1080     Conference_id int NOT NULL IDENTITY,
1081     name varchar(100) NOT NULL,
1082     description text NULL,
1083     CONSTRAINT Conferences_pk PRIMARY KEY (Conference_id)
1084 );
1085
1086 — Table: Early_signup_discounts
1087 CREATE TABLE Early_signup_discounts (
1088     discount_id int NOT NULL IDENTITY,
1089     conference_day_id int NOT NULL,
1090     end_date datetime NOT NULL,
1091     discount decimal(4,4) NOT NULL DEFAULT 0,
1092     CONSTRAINT Early_signup_discounts_pk PRIMARY KEY (discount_id)
1093 );
1094
1095 CREATE INDEX esd_end_date on Early_signup_discounts (end_date ASC)
1096 ;
1097
1098 — Table: Participants
1099 CREATE TABLE Participants (
1100     participant_id int NOT NULL IDENTITY,
1101     clients_id int NULL DEFAULT Null,
1102     name varchar(100) NOT NULL,
1103     surname varchar(100) NOT NULL,
1104     email varchar(100) NOT NULL CHECK (email like '%_@_%._%'),
1105     phone varchar(20) NOT NULL,
1106     CONSTRAINT Participants_pk PRIMARY KEY (participant_id)
1107 );
1108
1109 CREATE INDEX participants_client_id on Participants (clients_id ASC)
1110 ;
1111
1112 CREATE INDEX participants_name on Participants (surname ASC,name ASC)
1113 ;
1114
1115 — Table: Payments
1116 CREATE TABLE Payments (
1117     payment_id int NOT NULL IDENTITY,
1118     reservation_id int NOT NULL,
1119     in_date datetime NOT NULL,
1120     value money NOT NULL,
1121     CONSTRAINT Payments_pk PRIMARY KEY (payment_id)
1122 );
1123
1124 CREATE INDEX payment_reservation on Payments (reservation_id ASC)
1125 ;
1126
1127 — Table: Workshop_registration
1128 CREATE TABLE Workshop_registration (
1129     Participant_id int NOT NULL,
1130     reservation_id int NOT NULL,
1131     CONSTRAINT Workshop_registration_pk PRIMARY KEY (Participant_id,reservation_id)
1132 );
1133
1134 CREATE INDEX workshop_reg_part_id on Workshop_registration (Participant_id ASC)
1135 ;
1136
1137 CREATE INDEX workshop_reg_res_id on Workshop_registration (reservation_id ASC)
1138 ;
1139
1140 — Table: Workshop_reservations
1141 CREATE TABLE Workshop_reservations (
1142     reservation_id int NOT NULL IDENTITY,
1143     workshop_id int NOT NULL,
1144     reservation_date datetime NOT NULL DEFAULT GETDATE(),
1145     due_price datetime NOT NULL DEFAULT DATEADD(week, 2, GETDATE()) CHECK (due_price >=
1146     GETDATE()),
1147     nr_of_seats int NOT NULL DEFAULT 0 CHECK (nr_of_seats >= 0),

```

```

1148     Conference_day_res_id int NOT NULL,
active bit NOT NULL DEFAULT 1,
1150     CONSTRAINT Workshop_reservations_pk PRIMARY KEY (reservation_id)
);

1152 CREATE INDEX workshop_res_conf_res_id on Workshop_reservations (Conference_day_res_id ASC)
;
1154 CREATE INDEX workshop_res_pay_deadline on Workshop_reservations (due_price ASC)
1156 ;

1158 CREATE INDEX workshop_res_workshop_id on Workshop_reservations (workshop_id ASC)
;
1160
— Table: Workshops
1162 CREATE TABLE Workshops (
workshop_id int NOT NULL IDENTITY,
1164 conference_day_id int NOT NULL,
start_time datetime NOT NULL,
1166 end_time datetime NOT NULL CHECK (end_time >= GETDATE()),
topic varchar(100) NOT NULL,
1168 price money NOT NULL CHECK (price >= 0),
number_of_seats int NOT NULL DEFAULT 0 CHECK (number_of_seats >= 0),
1170 CONSTRAINT Workshops_pk PRIMARY KEY (workshop_id)
);
1172
CREATE INDEX workshop_day_id on Workshops (conference_day_id ASC)
1174 ;

1176 CREATE INDEX workshop_time on Workshops (start_time ASC,end_time ASC)
;
1178
— foreign keys
1180 — Reference: Companies_Clients (table: Companies)
ALTER TABLE Companies ADD CONSTRAINT Companies_Clients
1182 FOREIGN KEY (clients_id)
REFERENCES Clients (id);
1184
— Reference: Conference_day_registration_Conference_day_reservations (table:
Conference_day_registration)
1186 ALTER TABLE Conference_day_registration ADD CONSTRAINT
Conference_day_registration_Conference_day_reservations
FOREIGN KEY (reservation_id)
1188 REFERENCES Conference_day_reservations (reservation_id)
ON DELETE CASCADE;
1190
— Reference: Conference_day_registration_Participants (table: Conference_day_registration)
1192 ALTER TABLE Conference_day_registration ADD CONSTRAINT
Conference_day_registration_Participants
FOREIGN KEY (Participant_id)
1194 REFERENCES Participants (participant_id)
ON DELETE CASCADE;
1196
— Reference: Conference_day_reservations_Clients (table: Conference_day_reservations)
1198 ALTER TABLE Conference_day_reservations ADD CONSTRAINT Conference_day_reservations_Clients
FOREIGN KEY (clients_id)
1200 REFERENCES Clients (id);

1202 — Reference: Conference_day_reservations_Conference_days (table: Conference_day_reservations)
ALTER TABLE Conference_day_reservations ADD CONSTRAINT
Conference_day_reservations_Conference_days
1204 FOREIGN KEY (conference_day_id)
REFERENCES Conference_days (conference_day_id)
1206 ON DELETE CASCADE;

1208 — Reference: Conference_days_Conferences (table: Conference_days)
ALTER TABLE Conference_days ADD CONSTRAINT Conference_days_Conferences
1210 FOREIGN KEY (conference_id)
REFERENCES Conferences (Conference_id)
1212 ON DELETE CASCADE;

1214 — Reference: Discounts_Conference_days (table: Early_signup_discounts)
ALTER TABLE Early_signup_discounts ADD CONSTRAINT Discounts_Conference_days
1216 FOREIGN KEY (conference_day_id)
REFERENCES Conference_days (conference_day_id)
1218 ON DELETE CASCADE;

```

```

1220 — Reference: Participants_Clients (table: Participants)
1221 ALTER TABLE Participants ADD CONSTRAINT Participants_Clients
1222 FOREIGN KEY (clients_id)
1223 REFERENCES Clients (id)
1224 ON DELETE SET NULL;

1226 — Reference: Payments_Conference_day_reservations (table: Payments)
1227 ALTER TABLE Payments ADD CONSTRAINT Payments_Conference_day_reservations
1228 FOREIGN KEY (reservation_id)
1229 REFERENCES Conference_day_reservations (reservation_id);

1230 — Reference: Workshop_registration_Participants (table: Workshop_registration)
1231 ALTER TABLE Workshop_registration ADD CONSTRAINT Workshop_registration_Participants
1232 FOREIGN KEY (Participant_id)
1233 REFERENCES Participants (participant_id)
1234 ON DELETE CASCADE;

1236 — Reference: Workshop_registration_Workshop_reservations (table: Workshop_registration)
1237 ALTER TABLE Workshop_registration ADD CONSTRAINT Workshop_registration_Workshop_reservations
1238 FOREIGN KEY (reservation_id)
1239 REFERENCES Workshop_reservations (reservation_id)
1240 ON DELETE CASCADE;

1242 — Reference: Workshop_reservations_Conference_day_reservations (table: Workshop_reservations)
1243 ALTER TABLE Workshop_reservations ADD CONSTRAINT
1244 Workshop_reservations_Conference_day_reservations
1245 FOREIGN KEY (Conference_day_res_id)
1246 REFERENCES Conference_day_reservations (reservation_id)
1247 ON DELETE CASCADE;

1248 — Reference: Workshop_reservations_Workshops (table: Workshop_reservations)
1249 ALTER TABLE Workshop_reservations ADD CONSTRAINT Workshop_reservations_Workshops
1250 FOREIGN KEY (workshop_id)
1251 REFERENCES Workshops (workshop_id)
1252 ON DELETE CASCADE;

1254 — Reference: Workshops_Conference_days (table: Workshops)
1255 ALTER TABLE Workshops ADD CONSTRAINT Workshops_Conference_days
1256 FOREIGN KEY (conference_day_id)
1257 REFERENCES Conference_days (conference_day_id);

1260 — End of file.

```

../Create.sql

Widoki

```
1000 — Shows sum of payments per reservation
1001 CREATE VIEW PaymentsForReservation
1002 AS
1003 SELECT reservation_id , ISNULL(sum(Payments.value) , 0) as already_paid
1004 FROM Payments
1005 GROUP BY reservation_id
1006 GO

1008 — Show sum of payments per client
1009 CREATE VIEW PaymentsForClients
1010 AS
1011 SELECT clients_id , SUM(value) AS paid
1012 FROM Payments
1013         INNER JOIN Conference_day_reservations Cdr ON Payments.reservation_id = Cdr.
1014         reservation_id
1015 GROUP BY clients_id
1016 GO

1018 — Shows balance of payments per reservation
1019 CREATE VIEW BalanceForReservation
1020 AS
1021 SELECT reservation_id , already_paid - dbo.confReservationPrice(reservation_id) as balance
1022 FROM PaymentsForReservation
1023 GO

1024 — Shows payment balance per client
1025 CREATE VIEW ClientsBalanceView
1026 AS
1027 SELECT id                                as client_id ,
1028        dbo.clientsBalance(id) as balance
1029 FROM Clients
1030 GO

1032 — Shows clients with unpaid reservations
1033 CREATE VIEW ClientsWithUnpaidReservations
1034 AS
1035 SELECT clients_id ,
1036        reservation_id ,
1037        due_price
1038        as to_pay_until ,
1039        DATEDIFF(DAY, GETDATE() , due_price) as days_left ,
1040        dbo.confReservationPrice(reservation_id) - dbo.confReservationPaidAmount(reservation_id)
1041        ) as left_to_pay ,
1042        dbo.showClientsName(clients_id)
1043        as clients_name
1044 FROM Conference_day_reservations
1045 WHERE dbo.confReservationPrice(reservation_id) - dbo.confReservationPaidAmount(reservation_id)
1046        > 0.01
1047 and Conference_day_reservations.active = 1
1048 GO

1046 — Same but only for companies
1047 CREATE VIEW CompaniesWithUnpaidReservations
1048 AS
1049 SELECT *
1050 FROM ClientsWithUnpaidReservations
1051 WHERE dbo.isClientCompany(clients_id) = 1
1052 GO

1054 — Same but only for individual clients
1055 CREATE VIEW IndividualClientsWithUnpaidReservations
1056 AS
1057 SELECT *
1058 FROM ClientsWithUnpaidReservations
1059 WHERE dbo.isClientCompany(clients_id) = 0
1060 GO

1062 — Same, but with the earliest payment deadline
1063 CREATE VIEW UnpaidReservationsWithEarliestDeadline
1064 AS
1065 SELECT TOP 10 *
1066 FROM ClientsWithUnpaidReservations
1067 ORDER BY to_pay_until ASC
```

```

GO
1070
1071 — Clients with unpaid reservations who exceeded payment deadline
1072 CREATE VIEW ClientsWithExceededPaymentDeadline
1073 AS
1074     SELECT *
1075     FROM ClientsWithUnpaidReservations
1076     WHERE to_pay_until < GETDATE()
1077 GO
1078
1079 — Workshops with free seats available
1080 CREATE VIEW WorkshopsWithFreeSeats
1081 AS
1082 SELECT workshop_id ,
1083         dbo.workshopFreeSeats(workshop_id) as free_seats
1084 FROM Workshops
1085 WHERE dbo.workshopFreeSeats(workshop_id) > 0
1086 GO
1087
1088 — Conference days with free seats available
1089 CREATE VIEW ConferenceDaysWithFreeSeats
1090 AS
1091 SELECT conference_day_id ,
1092         dbo.conferenceFreeSeats(conference_day_id) as free_seats
1093 FROM Conference_days
1094 WHERE dbo.conferenceFreeSeats(conference_day_id) > 0
1095 GO
1096
1097 — Clients who paid the most
1098 CREATE VIEW ClientsWhoPaidMost
1099 AS
1100 SELECT TOP 20 *
1101 from PaymentsForClients
1102 ORDER BY paid DESC
1103 GO
1104
1105 — Clients with the highest count of conference reservations
1106 CREATE VIEW MostActiveClients
1107 AS
1108 SELECT TOP 20 clients_id , COUNT(reservation_id) AS reservation_count
1109 FROM Clients
1110     INNER JOIN Conference_day_reservations Cdr on Clients.id = Cdr.clients_id
1111 GROUP BY clients_id
1112 ORDER BY COUNT(reservation_id) DESC
1113 GO
1114
1115 — Show participants for conference days
1116 CREATE VIEW ParticipantsForConferenceDay
1117 AS
1118 SELECT name, surname, conference_day_id
1119 FROM Participants
1120     INNER JOIN Conference_day_registration Cdr on Participants.participant_id = Cdr.
1121     Participant_id
1122     INNER JOIN Conference_day_reservations Cdr on Cdr.reservation_id = Cdr.
1123     reservation_id
1124
1125 — Show participants for workshops
1126 CREATE VIEW ParticipantsForWorkshop
1127 AS
1128 SELECT name, surname, workshop_id
1129 FROM Participants
1130     INNER JOIN Workshop_registration Wrg on Participants.participant_id = Wrg.
1131     Participant_id
1132     INNER JOIN Workshop_reservations Wrs on Wrg.reservation_id = Wrs.reservation_id
1133
1134 — Show cancelled conference reservations
1135 CREATE VIEW CancelledConferenceReservations
1136 AS
1137 SELECT reservation_id ,
1138         conference_day_id ,
1139         reservation_date ,
1140         clients_id ,
1141         adult_seats ,
1142         student_seats ,
1143         dbo.showClientsName(clients_id) as clients_name
1144 FROM Conference_day_reservations

```



```

1142 WHERE active = 0

1144 — Show cancelled workshops reservations
CREATE VIEW CancelledWorkshopsReservations
1146 AS
    SELECT Workshop_reservations.reservation_id ,
1148           workshop_id ,
           Workshop_reservations.reservation_date ,
1150           nr_of_seats ,
           dbo.showClientsName(clients_id) as clients_name
1152 FROM Workshop_reservations
    INNER JOIN Conference_day_reservations Cdr on Workshop_reservations.
           Conference_day_res_id = Cdr.reservation_id
1154 WHERE Workshop_reservations.active = 0
GO

1156 — Show how popular conferences are
1158 CREATE VIEW ConferencePopularity
AS
1160 SELECT conf.Conference_id , conf.name, SUM(cdr.student_seats + cdr.adult_seats) AS
           participants_count
           FROM Conferences conf INNER JOIN Conference_days cd on conf.Conference_id = cd.
           conference_id
1162 INNER JOIN Conference_day_reservations cdr on cd.conference_day_id = cdr.conference_day_id
           WHERE cdr.active = 1
1164 GROUP BY conf.Conference_id , conf.name
GO

1166 — Show the most popular conferences
1168 CREATE VIEW MostPopularConference
AS
1170 SELECT TOP 20 *
           FROM ConferencePopularity
1172 ORDER BY participants_count DESC
GO

1174 — Show how popular Workshops are
1176 CREATE VIEW WorkshopPopularity
AS
1178 SELECT w.workshop_id , w.topic , SUM(wr.nr_of_seats) AS participants_count
           FROM Workshops w INNER JOIN Workshop_reservations wr on w.workshop_id = wr.workshop_id
1180 WHERE wr.active=1
           GROUP BY w.workshop_id , w.topic
1182 GO

1184 — Show the most popular workshops
1186 CREATE VIEW MostPopularWorkshops
AS
           SELECT TOP 20 *
           FROM WorkshopPopularity
1188 ORDER BY participants_count DESC
1190 GO

1192 — Show all cities from which our clients are coming
CREATE VIEW ClientsCities
1194 AS
           SELECT DISTINCT city
           FROM Clients
1196 GO

1198 — Show all clients phone numbers
1200 CREATE VIEW ClientsPhones
AS
1202 SELECT DISTINCT phone
           FROM Clients INNER JOIN Companies C on Clients.id = C.clients_id
1204 UNION
           SELECT DISTINCT phone
           FROM Clients INNER JOIN Participants P on Clients.id = P.clients_id
1206 GO

1208 — Show all clients phone numbers
1210 CREATE VIEW ClientsEmails
AS
1212 SELECT DISTINCT email
           FROM Clients INNER JOIN Companies C on Clients.id = C.clients_id
1214 UNION

```

```
1216 | SELECT DISTINCT email  
      | FROM Clients INNER JOIN Participants P on Clients.id = P.clients_id  
      | GO
```

../Views/Views.sql

Funkcje

Funkcja sprawdzająca czy wprowadzony NIP jest zgodny z obowiązującymi prawami.

```
1000 CREATE FUNCTION IsValidNip(  
1001     @nip nvarchar(15)  
1002 )  
1003     RETURNS bit  
1004 AS  
1005 BEGIN  
1006     IF ISNUMERIC(@nip) = 0  
1007         BEGIN  
1008             RETURN 0  
1009         END  
1010  
1011     IF @nip = '0000000000'  
1012         BEGIN  
1013             RETURN 0  
1014         END  
1015     IF @nip = '1234567891'  
1016         BEGIN  
1017             RETURN 0  
1018         END  
1019     IF @nip = '1111111111'  
1020         BEGIN  
1021             RETURN 0  
1022         END  
1023     IF @nip = '1111111112'  
1024         BEGIN  
1025             RETURN 0  
1026         END  
1027     IF @nip = '9999999999'  
1028         BEGIN  
1029             RETURN 0  
1030         END  
1031     IF @nip = '1111111112'  
1032         BEGIN  
1033             RETURN 0  
1034         END  
1035  
1036     DECLARE @sum INT;  
1037     SET @sum = 6 * CONVERT(INT, SUBSTRING(@nip, 1, 1)) +  
1038         5 * CONVERT(INT, SUBSTRING(@nip, 2, 1)) +  
1039         7 * CONVERT(INT, SUBSTRING(@nip, 3, 1)) +  
1040         2 * CONVERT(INT, SUBSTRING(@nip, 4, 1)) +  
1041         3 * CONVERT(INT, SUBSTRING(@nip, 5, 1)) +  
1042         4 * CONVERT(INT, SUBSTRING(@nip, 6, 1)) +  
1043         5 * CONVERT(INT, SUBSTRING(@nip, 7, 1)) +  
1044         6 * CONVERT(INT, SUBSTRING(@nip, 8, 1)) +  
1045         7 * CONVERT(INT, SUBSTRING(@nip, 9, 1));  
1046  
1047     IF CONVERT(TINYINT, SUBSTRING(@nip, 10, 1)) = (@sum % 11)  
1048         BEGIN  
1049             RETURN 1  
1050         END  
1051     RETURN 0  
1052 END
```

../Functions/IsValidNip.sql

Funkcja sprawdzająca czy dany klient jest firmą.

```
1000 CREATE FUNCTION isClientCompany(  
1001     @client_id int  
1002 )  
1003     RETURNS bit  
1004 AS  
1005 BEGIN  
1006     DECLARE @res bit = ISNULL((SELECT 1 FROM Companies WHERE clients_id=@client_id),0)  
1007     RETURN @res  
1008 END
```

../Functions/isClientCompany.sql

Funkcja zwracająca imię i nazwisko klienta (jeśli jest on osobą fizyczną).

```
1000 CREATE FUNCTION showClientsName(  
1001     @client_id int  
1002 )  
1003     RETURNS nvarchar(100)  
1004 AS  
1005 BEGIN  
1006     DECLARE @res nvarchar(100) = ISNULL((SELECT clients_name FROM Companies WHERE clients_id=@client_id),0)  
1007     RETURN @res  
1008 END
```

```

1002 )
1003     RETURNS varchar(100)
1004 AS
1005 BEGIN
1006     DECLARE @name VARCHAR(100) = (SELECT name + ' ' + surname FROM Participants WHERE
1007         clients_id = @client_id)
1008     if @name is not null
1009         BEGIN
1010             RETURN @name
1011         END
1012     SET @name = (SELECT companyName FROM Companies WHERE clients_id = @client_id)
1013     RETURN @name
1014 END

```

../Functions/showClientsName.sql

Funkcja zwracająca ilość wolnych miejsc na danej konferencji.

```

1000 CREATE FUNCTION conferenceFreeSeats(@conferenceId int)
1001     RETURNS INT
1002 AS
1003 BEGIN
1004     DECLARE @used INT
1005     SET @used = ISNULL((SELECT SUM(adult_seats + student_seats)
1006         FROM Conference_day_reservations
1007         WHERE conference_day_id = @conferenceId), 0)
1008     DECLARE @all INT
1009     SET @all = (SELECT number_of_seats
1010         FROM Conference_days
1011         WHERE conference_day_id = @conferenceId)
1012     RETURN @all - @used
1013 END
1014 GO

```

../Functions/FreeSeats/conferenceFreeSeats.sql

Funkcja zwracająca ilość wolnych miejsc na danych warsztatach.

```

1000 CREATE FUNCTION workshopFreeSeats(@workshopId int)
1001     RETURNS INT
1002 AS
1003 BEGIN
1004     DECLARE @used INT
1005     SET @used = ISNULL((SELECT SUM(nr_of_seats)
1006         FROM Workshop_reservations
1007         WHERE workshop_id = @workshopId), 0)
1008     DECLARE @all INT
1009     SET @all = (SELECT number_of_seats
1010         FROM Workshops
1011         WHERE workshop_id = @workshopId)
1012     RETURN @all - @used
1013 END
1014 GO

```

../Functions/FreeSeats/workshopFreeSeats.sql

Funkcja zwracająca imiona i nazwiska uczestników danej konferencji.

```

1000 CREATE FUNCTION participantsForGivenConference(
1001     @conferenceID int
1002 )
1003     RETURNS table
1004 AS
1005     RETURN SELECT name, surname
1006         FROM ParticipantsForConferenceDay
1007         WHERE conference_day_id = @conferenceID

```

../Functions/ParticipantsLists/participantsForGivenConference.sql

Funkcja zwracająca imiona i nazwiska uczestników danego warsztatu.

```

1000 CREATE FUNCTION participantsForGivenWorkshop(
1001     @workshopID int
1002 )
1003     RETURNS table
1004 AS

```

```

1006      RETURN SELECT name, surname
      FROM ParticipantsForWorkshop
      WHERE workshop_id = @workshopID

```

../Functions/ParticipantsLists/participantsForGivenWorkshop.sql

Funkcja zwracająca całkowity balans danego klienta.

```

1000 CREATE FUNCTION clientsBalance(@ClientId int)
      RETURNS MONEY
1002 AS
      BEGIN
1004         DECLARE @paid MONEY = (SELECT SUM(P.VALUE)
                                FROM Payments P
                                inner join Conference_day_reservations Cdr on P.
                                reservation_id = Cdr.reservation_id
                                WHERE Cdr.clients_id = @ClientId and cdr.active = 1)
1008         DECLARE @owed MONEY = (SELECT SUM(dbo.confReservationPrice(cdr.reservation_id))
                                FROM Conference_day_reservations cdr
                                WHERE cdr.clients_id = @ClientId and cdr.active = 1)
1010         DECLARE @res MONEY = ROUND(ISNULL(@paid, 0) - ISNULL(@owed, 0), 2);
1012         IF @res = 0.01
            BEGIN
1014             return 0
            END
1016         RETURN @res
      END

```

../Functions/PaymentsAndPrices/clientsBalance.sql

Funkcja zwracająca koszt danej rezerwacji na dzień konferencji (łącznie z ceną podpiętych warsztatów).

```

1000 CREATE FUNCTION confDayPrice(@reservationId int)
      RETURNS MONEY
1002 AS
      BEGIN
1004         DECLARE @earlySignupDiscount decimal(4, 4) = ISNULL((Select TOP 1 esd.discount
                                                                from Conference_day_reservations as
                                                                res
                                                                inner join Conference_days
                                                                as day on res.conference_day_id = day.conference_day_id
                                                                inner join
                                                                Early_signup_discounts esd
                                                                on day.
                                                                conference_day_id = esd.conference_day_id
                                                                where reservation_id = @reservationId
                                                                and esd.end_date > res.
                                                                reservation_date
                                                                order by esd.end_date ASC), 0)
1012         DECLARE @nrOfStudents int = (SELECT student_seats
                                         FROM Conference_day_reservations
                                         WHERE reservation_id = @reservationId)
1014         DECLARE @nrOfAdults int = (SELECT adult_seats
                                      FROM Conference_day_reservations
                                      WHERE reservation_id = @reservationId)
1016         DECLARE @conferenceDayID int = (SELECT conference_day_id
                                           FROM Conference_day_reservations
                                           WHERE reservation_id = @reservationId)
1018         DECLARE @standardPrice money = (SELECT standard_price
                                           FROM Conference_days
                                           WHERE conference_day_id = @conferenceDayID)
1020         DECLARE @studentDiscount decimal(4, 4) = (SELECT student_discount
                                                    FROM Conference_days
                                                    WHERE conference_day_id = @conferenceDayID)
1022         DECLARE @price money = (@nrOfAdults * @standardPrice) +
1024                                (@nrOfStudents * (@standardPrice * (1.0 - @studentDiscount)))
1026         RETURN CONVERT(money, ROUND(@price * (1.0 - @earlySignupDiscount), 2))
1030 END
      GO

```

../Functions/PaymentsAndPrices/confDayPrice.sql

Funkcja zwracająca dotychczasowo wpłaconą kwotę na daną rezerwację.

```

1000 CREATE FUNCTION confReservationPaidAmount(@reservation_id int)
      RETURNS MONEY
1002 AS

```

```

BEGIN
1004 RETURN ISNULL((SELECT sum(value) FROM Payments WHERE reservation_id = @reservation_id), 0)
END

```

../Functions/PaymentsAndPrices/confReservationPaidAmount.sql

Funkcja zwracająca koszt danej rezerwacji na warsztaty.

```

1000 CREATE FUNCTION confReservationPrice(@reservation_id int)
      RETURNS MONEY
1002 AS
      BEGIN
1004 DECLARE @dayPrice money = dbo.confDayPrice(@reservation_id);
      DECLARE @workshopsPrice money = (select sum(wr.nr_of_seats * W.price)
1006                                     from Workshop_reservations wr
                                     inner join Workshops W on wr.workshop_id = W.
      workshop_id
1008                                     where wr.Conference_day_res_id = @reservation_id);
1010 RETURN ROUND(@dayPrice + ISNULL(@workshopsPrice, 0), 2)
END

```

../Functions/PaymentsAndPrices/confReservationPrice.sql

Procedury

Procedury służące wprowadzaniu nowych danych do bazy:

```
1000 — Add client to database
1001 CREATE PROCEDURE AddClientParticipant @ZipCode varchar(6), @City varchar(30), @Address varchar
      (100), @Name varchar(100),
1002                                     @Surname varchar(100),
                                     @Email varchar(100), @Phone varchar(20)
1003
1004 AS
1005 BEGIN
1006     SET NOCOUNT ON;
1007     INSERT INTO Clients(zip_code, city, address)
1008     VALUES (@ZipCode, @City, @Address)
1009
1010     INSERT INTO Participants (clients_id, name, surname, email, phone)
1011     VALUES (SCOPE_IDENTITY(), @Name, @Surname, @Email, @Phone)
1012 END
1013 GO
1014
1015 CREATE PROCEDURE AddClientCompany @ZipCode varchar(6), @City varchar(30), @Address varchar
      (100),
1016                                     @companyName varchar(100), @nip nvarchar(15), @phone varchar
      (20), @email varchar(100)
1017
1018 AS
1019 BEGIN
1020     SET NOCOUNT ON;
1021     INSERT INTO Clients(zip_code, city, address)
1022     VALUES (@ZipCode, @City, @Address)
1023
1024     INSERT INTO Companies (companyName, nip, phone, clients_id, email)
1025     VALUES (@companyName, @nip, @phone, scope_identity(), @email)
1026 END
1027 GO
1028
1029 — Add participant
1030 CREATE PROCEDURE AddParticipant @Name varchar(100), @Surname varchar(100),
      @Email varchar(100), @Phone varchar(20)
1031
1032 AS
1033 BEGIN
1034     SET NOCOUNT ON;
1035     INSERT INTO Participants(name, surname, email, phone)
1036     VALUES (@Name, @Surname, @Email, @Phone)
1037 END
1038 GO
1039
1040 CREATE PROCEDURE AddPayment @ReservationID int, @Value money
1041
1042 AS
1043 BEGIN
1044     SET NOCOUNT ON;
1045     INSERT INTO Payments(reservation_id, value)
1046     VALUES (@ReservationID, @Value) — date = getDate() by default
1047 END
1048 GO
1049
1050 CREATE PROCEDURE AddConference @Name varchar(100),
      @description text = null
1051
1052 AS
1053 BEGIN
1054     SET NOCOUNT ON;
1055     INSERT INTO Conferences(name, description)
1056     VALUES (@Name, @description)
1057 END
1058 GO
1059
1060 CREATE PROCEDURE AddConferenceDay @ConferenceId int,
      @Date date,
      @StandardPrice money,
      @StudentDiscount int,
      @NumberOfSeats int
1061
1062 AS
1063 BEGIN
1064     SET NOCOUNT ON;
1065     INSERT INTO Conference_days
      (conference_id, date, standard_price, student_discount, number_of_seats)
1066     VALUES (@ConferenceId, @Date, @StandardPrice, @StudentDiscount, @NumberOfSeats)
1067 END
```

```

1070 GO
1072 CREATE PROCEDURE AddConferenceWithDays @Name varchar(100),
1073                                     @numberOfDays int,
1074                                     @startDate date,
1075                                     @price money,
1076                                     @studentDiscount decimal(4, 4),
1077                                     @numberOfSeats int,
1078                                     @description text = null
1079 AS
1080 BEGIN
1081     SET NOCOUNT ON;
1082     INSERT INTO Conferences(name, description)
1083     VALUES (@Name, @description)
1084
1085     DECLARE @confId int = SCOPE_IDENTITY()
1086
1087     WHILE @numberOfDays > 0
1088     BEGIN
1089         EXEC AddConferenceDay
1090             @confId,
1091             @startDate,
1092             @price,
1093             @studentDiscount,
1094             @numberOfSeats;
1095         SET @startDate = DATEADD(DAY, 1, @startDate)
1096         SET @numberOfSeats = @numberOfSeats - 1
1097     END
1098 END
1099 GO
1100
1101 CREATE PROCEDURE AddEarlySignupDiscount @ConferenceDayId int,
1102                                     @EndDate datetime,
1103                                     @Discount decimal(4, 4)
1104 AS
1105 BEGIN
1106     SET NOCOUNT ON;
1107     INSERT INTO Early_signup_discounts(conference_day_id, end_date, discount)
1108     VALUES (@ConferenceDayId, @EndDate, @Discount)
1109 END
1110 GO
1111
1112 CREATE PROCEDURE AddEarlySignupDiscountToAllInConf @ConferenceId int,
1113                                     @EndDate datetime,
1114                                     @Discount decimal(4, 4)
1115 AS
1116 BEGIN
1117     SET NOCOUNT ON;
1118     CREATE TABLE #ids
1119     (
1120         rn int,
1121         id int
1122     )
1123     INSERT INTO #ids
1124     SELECT DISTINCT row_number() over (order by conference_day_id) as rn, conference_day_id as
1125         id
1126     FROM Conference_days
1127     WHERE conference_id = @ConferenceId;
1128
1129     DECLARE @id int
1130     DECLARE @totalrows int = (select count(*) from #ids)
1131     DECLARE @currentrow int = 1
1132
1133     WHILE @currentrow <= @totalrows
1134     BEGIN
1135         set @id = (select id from #ids where rn = @currentrow)
1136         exec AddEarlySignupDiscount
1137             @id,
1138             @EndDate,
1139             @Discount
1140         set @currentrow = @currentrow + 1
1141     END
1142 END
1143 GO
1144
1145 CREATE PROCEDURE AddWorkshop @conference_day_id int,

```



```

1146         @start_time datetime,
1147         @end_time datetime,
1148         @topic varchar(100),
1149         @price money,
1150         @number_of_seats int
1151 AS
1152 BEGIN
1153     SET NOCOUNT ON;
1154     INSERT INTO Workshops(conference_day_id, start_time, end_time, topic, price,
1155         number_of_seats)
1156     VALUES (@conference_day_id, @start_time, @end_time, @topic, @price, @number_of_seats)
1157 END
1158 GO
1159
1160 CREATE PROCEDURE RegisterParticipantForConferenceDay @reservation_id int, @Participant_id int,
1161     @is_student bit
1162 AS
1163 BEGIN
1164     SET NOCOUNT ON;
1165     INSERT INTO Conference_day_registration(reservation_id, Participant_id, is_student)
1166     VALUES (@reservation_id, @Participant_id, @is_student)
1167 end
1168 go
1169
1170 CREATE PROCEDURE RegisterParticipantForWorkshop @reservation_id int, @Participant_id int
1171 AS
1172 BEGIN
1173     SET NOCOUNT ON;
1174     INSERT INTO Workshop_registration(reservation_id, Participant_id)
1175     VALUES (@reservation_id, @Participant_id)
1176 end
1177 go
1178
1179 CREATE PROCEDURE AddConferenceDayReservation @conference_day_id int,
1180     @clients_id int,
1181     @reservation_date datetime,
1182     @due_price datetime,
1183     @adult_seats int,
1184     @student_seats int
1185 AS
1186 BEGIN
1187     SET NOCOUNT ON;
1188     IF ((SELECT COUNT(conference_day_id) FROM Conference_days WHERE conference_day_id =
1189         @conference_day_id) = 0)
1190     BEGIN
1191         THROW 52000, 'There is no such conference day in database', 1;
1192     END
1193     INSERT INTO Conference_day_reservations(conference_day_id, clients_id, reservation_date,
1194         due_price,
1195         adult_seats, student_seats)
1196     VALUES (@conference_day_id, @clients_id, @reservation_date, @due_price, @adult_seats,
1197         @student_seats)
1198 end
1199 go
1200
1201 CREATE PROCEDURE AddConferenceReservation @conference_id int,
1202     @clients_id int,
1203     @reservation_date datetime,
1204     @due_price datetime,
1205     @adult_seats int,
1206     @student_seats int
1207 AS
1208 BEGIN
1209     SET NOCOUNT ON;
1210     IF ((SELECT COUNT(conference_id) FROM Conferences WHERE Conference_id = @conference_id) =
1211         0)
1212     BEGIN
1213         THROW 52000, 'There is no such conference in database', 1;
1214     END
1215
1216     CREATE TABLE #ids
1217     (
1218         rn int,
1219         id int
1220     )

```

```

1216     INSERT INTO #ids
1217     SELECT DISTINCT row_number() over (order by conference_day_id) as rn, conference_day_id as
1218     id
1219     FROM Conference_days
1220     WHERE conference_id = @conference_id;

1221
1222     DECLARE @id int
1223     DECLARE @totalrows int = (select count(*) from #ids)
1224     IF @totalrows=0
1225         BEGIN
1226             THROW 52000,'This conference has no days defined', 1;
1227         END
1228     DECLARE @currentrow int = 1

1229     WHILE @currentrow <= @totalrows
1230     BEGIN
1231         SET @id = (select id from #ids where rn = @currentrow)
1232         EXEC AddConferenceDayReservation
1233             @id,
1234             @clients_id,
1235             @reservation_date,
1236             @due_price,
1237             @adult_seats,
1238             @student_seats
1239         SET @currentrow = @currentrow + 1
1240     END

1241 END
1242 GO

1243
1244 CREATE PROCEDURE AddWorkshopReservation @workshop_id int,
1245                                         @conf_reservation_id int,
1246                                         @nr_of_seats int,
1247                                         @weeks_to_pay int
1248 AS
1249 BEGIN
1250     SET NOCOUNT ON;
1251     IF ((SELECT COUNT(workshop_id) FROM Workshops WHERE workshop_id = @workshop_id) = 0)
1252         BEGIN
1253             THROW 52000,'There is no such workshop in database', 1;
1254         END

1255     INSERT INTO Workshop_reservations(workshop_id, Conference_day_res_id, nr_of_seats,
1256     due_price)
1257     VALUES (@workshop_id, @conf_reservation_id, @nr_of_seats, DATEADD(week, @weeks_to_pay,
1258     GETDATE()))
1259 END
1260 GO

```

../Procedures/AdditionProcedures.sql

Procedury usuwające dane z bazy:

```

1000 CREATE PROCEDURE DeleteWorkshopRegistration @ParticipantID int, @ReservationID int
1001 AS
1002 BEGIN
1003     SET NOCOUNT ON;
1004     BEGIN TRY
1005         BEGIN TRAN DELETE
1006             FROM Workshop_registration
1007             WHERE reservation_id = @ReservationId
1008             AND Participant_id = @ParticipantID
1009         COMMIT TRAN
1010     END TRY
1011     BEGIN CATCH
1012         PRINT error_message() ROLLBACK TRANSACTION
1013     END CATCH
1014 END
1015 GO

1016
1017 CREATE PROCEDURE DeleteconferenceRegistration @ParticipantID int, @ReservationID int
1018 AS
1019 BEGIN
1020     SET NOCOUNT ON;
1021     BEGIN TRY
1022         BEGIN TRAN DELETE

```

```

1024         FROM Conference_day_registration
        WHERE reservation_id = @ReservationId
        AND Participant_id = @ParticipantID
1026     COMMIT TRAN
1028 END TRY
1028 BEGIN CATCH
        PRINT error_message() ROLLBACK TRANSACTION
1030 END CATCH
1032 END
GO

```

../Procedures/DeleteProcedures.sql

Procedury aktualizujące informacje zawarte w bazie danych:

```

1000 — procedura pozwalająca zmniejszyć ilość zarezerwowanych miejsc na warsztaty, pod
1001 warunkiem że na daną rezerwację
1002 — nie zarejestrowało się już więcej osób
1002 CREATE PROCEDURE DecreaseNumberOfBookedPlacesForWorkshop @reservation_id int, @new_nr_of_seats
1003 int
1004 AS
1004 BEGIN
1005     SET NOCOUNT ON;
1006     DECLARE @already_registered_participants int = (SELECT COUNT(*)
1007                                                     FROM Workshop_registration
1008                                                     WHERE reservation_id = @reservation_id)
1009     IF @new_nr_of_seats < (SELECT nr_of_seats FROM Workshop_reservations WHERE reservation_id
1010 = @reservation_id)
1011     BEGIN
1012         IF (@already_registered_participants <= @new_nr_of_seats)
1013         BEGIN
1014             UPDATE Workshop_reservations
1015             SET nr_of_seats = @new_nr_of_seats
1016             WHERE reservation_id = @reservation_id
1017             PRINT 'success'
1018         END
1019     ELSE
1020     BEGIN
1021         PRINT 'cannot decrease number of booked places, as ' +
1022             CAST(@already_registered_participants AS varchar(4)) +
1023             ' participants already registered for the workshop on this
1024 reservation'
1025     END
1026 END
1027 GO
1028 — procedura dezaktywująca rezerwację warsztatu
1028 CREATE PROCEDURE DeactivateWorkshopReservation @reservation_id int
1029 AS
1030 BEGIN
1031     SET NOCOUNT ON;
1032     DECLARE @currentlyActive bit = (SELECT active FROM Workshop_reservations WHERE
1033 reservation_id = @reservation_id)
1034     IF @currentlyActive = 1
1035     BEGIN
1036         UPDATE Workshop_reservations
1037         SET active = 0
1038         WHERE reservation_id = @reservation_id
1039     END
1040 END
1041 GO
1042 — procedura dezaktywująca rezerwację konferencji
1042 CREATE PROCEDURE DeactivateConferenceReservation @reservation_id int
1043 AS
1044 BEGIN
1045     SET NOCOUNT ON;
1046     DECLARE @currentlyActive bit = (SELECT active FROM Conference_day_reservations WHERE
1047 reservation_id = @reservation_id)
1048     IF @currentlyActive = 1
1049     BEGIN
1050         UPDATE Conference_day_reservations
1051         SET active = 0
1052         WHERE reservation_id = @reservation_id
1053     END
1054 END

```

1056 | GO

../Procedures/UpdateProcedures.sql

Triggery

```
1000  ---
1001  --- Triggery dla tabeli Conference_days
1002  ---
1003  --- sprawdzenie czy na dan konferencje nie zosta y dodane dwa te same dni konferencji
1004  CREATE TRIGGER CheckForTwoTheSameConferenceDays
1005  ON Conference_days
1006  AFTER INSERT, UPDATE AS
1007  BEGIN
1008  DECLARE @Date date = (SELECT date FROM inserted)
1009  DECLARE @ConferenceId int = (SELECT conference_id FROM inserted)
1010  IF ((SELECT COUNT(conference_day_id)
1011      FROM Conference_days
1012      WHERE (date = @Date)
1013           AND (conference_id = @ConferenceId)) > 1)
1014  BEGIN
1015      DECLARE @message varchar(100) = 'Day ' + CAST(@Date as varchar(11)) +
1016      ' has already been added for this conference';
1017      THROW 52000, @message, 1 ROLLBACK TRANSACTION
1018  END
1019  END
1020  ---
1021  --- Triggery dla tabeli Conference_day_reservations
1022  ---
1023  --- przy rzerwacji miejsc na konferencje sprawdza czy jest odpowiednia ilo wolnych miejsc
1024  CREATE TRIGGER CheckIfEnoughSeatsAvailableForConference
1025  ON Conference_day_reservations
1026  AFTER INSERT, UPDATE AS
1027  BEGIN
1028  DECLARE @ConferenceDayID int = (SELECT conference_day_id FROM inserted)
1029  DECLARE @RequestedSeats int = (SELECT (adult_seats + student_seats) FROM inserted)
1030  DECLARE @FreeSeats int = (SELECT [dbo].[conferenceFreeSeats](@ConferenceDayID))
1031  IF (@FreeSeats < @RequestedSeats)
1032  BEGIN
1033      DECLARE @Message varchar(100) = 'There are only ' + CAST(@FreeSeats as varchar(10))
1034      ' places left for this conference day.';
1035      THROW 52000, @Message, 1 ROLLBACK TRANSACTION
1036  END
1037  END
1038  ---
1039  --- Triggery dla tabeli Conference_day_registration
1040  ---
1041  --- trigger zapewniaj cy aby na dan rezerwacj dnia konferencji nie zapisa o si wi cej
1042  --- os b
1043  --- ni jest zarezerwowanych miejsc
1044  CREATE TRIGGER CheckIfConfReservationNotFull
1045  ON Conference_day_registration
1046  AFTER INSERT, UPDATE AS
1047  BEGIN
1048  DECLARE @reservationId int = (SELECT reservation_id FROM inserted)
1049  DECLARE @isStudent bit = (SELECT is_student FROM inserted)
1050  DECLARE @seatsTaken int = (SELECT COUNT(participant_id)
1051      FROM Conference_day_registration
1052      WHERE reservation_id = @reservationId
1053           AND is_student = @isStudent)
1054  DECLARE @seatsFree int = (SELECT IIF(@isStudent = 1, student_seats, adult_seats)
1055      FROM Conference_day_reservations
1056      WHERE reservation_id = @reservationId)
1057  IF (@seatsTaken > @seatsFree)
1058  BEGIN
1059      DECLARE @message varchar(100) = 'nr of participants for this reservation is
1060      exceeded, ' +
1061      CAST(@seatsTaken as varchar(3)) +
1062      ' participants are already registered';
1063      THROW 52000, @message, 1 ROLLBACK TRANSACTION
1064  END
1065  END
1066  --- sprawdzenie czy uczestnik zapisuje si na aktywn rezerwacj
1067  CREATE TRIGGER CheckIfCReservationActive
1068  ON Conference_day_registration
1069  AFTER INSERT, UPDATE AS
1070  BEGIN
```

```

1072 DECLARE @ReservationId int = (SELECT reservation_id FROM inserted)
1073 IF ((SELECT active FROM Conference_day_reservations WHERE reservation_id = @ReservationId)
1074     = 0)
1075     BEGIN
1076         DECLARE @message varchar(100) = 'corresponding reservation is not active';
1077         THROW 52000, @message, 1 ROLLBACK TRANSACTION
1078     END
1079 END

```

../Triggers/Conferences.sql

```

1000 -- =====
1001 -- Triggery dla tabeli Workshops
1002 -- =====
1003 -- sprawdzenie czy godzina rozpoczcia warsztatow jest wczeniejsza niz godzina jego
1004 -- zakonczenia
1005 CREATE TRIGGER StartHourLtEndHour
1006 ON Workshops
1007 AFTER INSERT, UPDATE AS
1008 BEGIN
1009     DECLARE @StartTime date = (SELECT start_time FROM inserted)
1010     DECLARE @EndTime date = (SELECT end_time FROM inserted)
1011     IF (@StartTime > @EndTime)
1012     BEGIN
1013         DECLARE @message varchar(100) = 'Workshop cant begin after it ends.';
1014         THROW 52000, @message, 1 ROLLBACK TRANSACTION
1015     END
1016 END
1017
1018 -- sprawdzenie czy tworzone warsztaty nie maj wikszej ilosci miejsc niz to ogranicza
1019 -- odpowiadajacy dzien konferencji
1020 CREATE TRIGGER WorkshopPlacesLtConfPlaces
1021 ON Workshops
1022 AFTER INSERT, UPDATE AS
1023 BEGIN
1024     DECLARE @confDayId int = (SELECT conference_day_id FROM inserted)
1025     DECLARE @nrOfseatsW int = (SELECT number_of_seats FROM inserted)
1026     DECLARE @nrOfseatsC int = (SELECT number_of_seats FROM Conference_days WHERE
1027         conference_day_id = @confDayId)
1028     IF (@nrOfseatsW > @nrOfseatsC)
1029     BEGIN
1030         DECLARE @message varchar(100) = 'Conference day offers less seats than proposed
1031         workshop';
1032         THROW 52000, @message, 1 ROLLBACK TRANSACTION
1033     END
1034 END
1035
1036 -- =====
1037 -- Triggery dla tabeli Workshop_reservations
1038 -- =====
1039 -- przy rezerwacji miejsc na warsztaty sprawdza czy jest odpowiednia ilosc wolnych miejsc na
1040 -- warsztatach
1041 CREATE TRIGGER CheckIfEnoughSeatsAvailableForWorkshop
1042 ON Workshop_reservations
1043 AFTER INSERT, UPDATE AS
1044 BEGIN
1045     DECLARE @WorkshopID int = (SELECT workshop_id FROM inserted)
1046     DECLARE @RequestedSeats int = (SELECT nr_of_seats FROM inserted)
1047     DECLARE @FreeSeats int = (SELECT [dbo].[workshopFreeSeats](@WorkshopID))
1048     IF (@FreeSeats < @RequestedSeats)
1049     BEGIN
1050         DECLARE @Message varchar(100) = 'There are only ' + CAST(@FreeSeats as varchar(10))
1051         +
1052         ' places left for this conference day.';
1053         THROW 52000, @Message, 1 ROLLBACK TRANSACTION
1054     END
1055 END
1056
1057 -- sprawdzenie czy klient nie probuje zarezerwowa warsztatu odbywajacego si w inny
1058 -- dzien
1059 -- ni moze byc zarezerwowany warsztat na ten sam dzien konferencji
1060 CREATE TRIGGER CheckIfWorkshopAndConfOnSameDay
1061 ON Workshop_reservations
1062 AFTER INSERT, UPDATE AS
1063 BEGIN

```

```

1058 DECLARE @WorkshopID int = (SELECT workshop_id FROM inserted)
1059 DECLARE @WorkshopDate date = (SELECT CONVERT(date, start_time)
1060                                FROM Workshops
1061                                WHERE Workshops.workshop_id = @WorkshopID)
1062 DECLARE @ConfDayResID int = (SELECT Conference_day_res_id FROM inserted)
1063 DECLARE @ConfDayID int = (SELECT conference_day_id
1064                            FROM Conference_day_reservations
1065                            WHERE reservation_id = @ConfDayResID)
1066 DECLARE @ConfDayDate date = (SELECT date FROM Conference_days WHERE conference_day_id =
@ConfDayID)
1067 IF (@ConfDayDate <> @WorkshopDate)
1068 BEGIN
1069     DECLARE @Message varchar(100) = 'workshop date differs from the associated
conference day date';
1070     THROW 52000, @Message, 1 ROLLBACK TRANSACTION
1071 END
1072 END
1073
1074 -- sprawdzenie czy odpowiadaj ca rezerwacji warsztatu rezerwacja dnia konferencji jest
aktywna
1075 CREATE TRIGGER CheckIfCorrespConfReservationActive
1076 ON Workshop_reservations
1077 AFTER INSERT, UPDATE AS
1078 BEGIN
1079     DECLARE @ConfResID int = (SELECT Conference_day_res_id FROM inserted)
1080     IF ((SELECT active FROM Conference_day_reservations cdr WHERE cdr.reservation_id =
@ConfResID) = 0)
1081     BEGIN
1082         DECLARE @Message varchar(100) = 'Corresponding conference day reservation is not
active';
1083         THROW 52000, @Message, 1 ROLLBACK TRANSACTION
1084     END
1085 END
1086
1087 -- procedura usuwaj ca uczestnik w warsztat w zapisanych na dezaktywowan rezerwacj
1088 CREATE TRIGGER OnDeactivateWReservation
1089 ON Workshop_reservations
1090 AFTER UPDATE AS
1091 BEGIN
1092     DECLARE @active bit = (SELECT active FROM inserted)
1093     DECLARE @ReservationID int = (SELECT reservation_id FROM inserted)
1094     IF (@active = 0)
1095     BEGIN TRY
1096         BEGIN TRAN DELETE
1097             FROM Workshop_registration
1098             WHERE reservation_id = @ReservationID
1099         COMMIT TRAN
1100     END TRY
1101     BEGIN CATCH
1102         PRINT error_message() ROLLBACK TRANSACTION
1103     END CATCH
1104 END
1105
1106 =====
1107 -- Triggery dla tabeli Workshop_registration
1108 =====
1109
1110 -- sprawdzenie czy uczestnik zapisuj cy si na warsztaty jest ju zapisany na odpowiedni
dzie konferencji
1111 CREATE TRIGGER CheckIfRegisteredForTheDay
1112 ON Workshop_registration
1113 AFTER INSERT, UPDATE AS
1114 BEGIN
1115     DECLARE @ReservationID int = (SELECT reservation_id FROM inserted)
1116     DECLARE @ParticipantID int = (SELECT participant_id FROM inserted)
1117     DECLARE @ConferenceDayID int = (SELECT conference_day_id
1118                                     FROM Workshops
1119                                     INNER JOIN Workshop_reservations
1120                                     on Workshops.workshop_id =
Workshop_reservations.workshop_id
1121                                     WHERE Workshop_reservations.reservation_id =
@ReservationID)
1122     IF (@ParticipantID not in (SELECT participant_id
1123                               FROM Conference_day_registration Cdr
1124                               INNER JOIN Conference_day_reservations C on Cdr.
reservation_id = C.reservation_id
1125                               WHERE C.conference_day_id = @ConferenceDayID))
1126     BEGIN

```

```

1126 DECLARE @message varchar(100) = 'Participant nr ' + CAST(@ParticipantID as varchar
(3)) +
' is not registered for the corresponding
conference day';
1128 THROW 52000,@message,1 ROLLBACK TRANSACTION
END
1130
-- sprawdzenie czy uczestnik zapisuj cy si na warsztaty nie jest ju zapisany na inne
warsztaty
1132 -- odbywaj ce si w tym samym czasie
CREATE TRIGGER CheckIfNotRegisteredForOtherWorkshop
1134 ON Workshop_registration
AFTER INSERT, UPDATE AS
1136 BEGIN
DECLARE @ReservationId int = (SELECT reservation_id FROM inserted)
1138 DECLARE @ParticipantID int = (SELECT participant_id FROM inserted)
DECLARE @StartTime datetime = (SELECT start_time
1140 FROM Workshops
INNER JOIN Workshop_reservations Wr on Workshops.
workshop_id = Wr.workshop_id
1142 WHERE Wr.reservation_id = @ReservationId)
DECLARE @EndTime datetime = (SELECT end_time
1144 FROM Workshops
INNER JOIN Workshop_reservations Wr on Workshops.
workshop_id = Wr.workshop_id
1146 WHERE Wr.reservation_id = @ReservationId)
DECLARE @CollidingWorkshops int = (SELECT Count(W.workshop_id)
1148 FROM Workshop_reservations Wrs
INNER JOIN Workshop_registration Wrg
1150 on Wrs.reservation_id = Wrg.
reservation_id AND
Wrg.Participant_id =
@ParticipantID
1152 INNER JOIN Workshops W on Wrs.workshop_id = W.
workshop_id
WHERE ((W.start_time BETWEEN @StartTime AND @EndTime)
1154 OR (W.end_time BETWEEN @StartTime AND @EndTime)))
IF (@CollidingWorkshops
1156 > 1) -- already registered for the day
BEGIN
1158 DECLARE @message varchar(100) = 'Participant nr ' + CAST(@ParticipantID as varchar
(3)) +
' already registered for ' + CAST(
@CollidingWorkshops as varchar(3)) +
1160 ' workshops at this time';
THROW 52000,@message,1 ROLLBACK TRANSACTION
1162 END
END
1164
-- trigger zapewniaj cy aby na dan rezerwacj warsztat w nie zapisa o si wi cej os b
ni jest zarezerwowanych miejsc
1166 CREATE TRIGGER CheckIfWorkshopReservationNotFull
ON Workshop_registration
1168 AFTER INSERT, UPDATE AS
BEGIN
1170 DECLARE @ReservationId int = (SELECT reservation_id FROM inserted)
DECLARE @PartForReserv int = (SELECT COUNT(Participant_id)
1172 FROM Workshop_registration Wr
WHERE Wr.reservation_id = @ReservationId)
1174 DECLARE @ReservedSeats int = (SELECT nr_of_seats FROM Workshop_reservations WHERE
reservation_id = @ReservationId)
1176 IF (@PartForReserv > @ReservedSeats)
BEGIN
1178 DECLARE @message varchar(100) = 'nr of participants for this reservation is
exceeded, ' +
CAST(@PartForReserv as varchar(3)) +
1180 ' participants are already registered';
THROW 52000,@message,1 ROLLBACK TRANSACTION
1182 END
END
1184
-- sprawdzenie czy uczestnik zapisuje si na aktywn rezerwacj
1186 CREATE TRIGGER CheckIfWReservationActive
ON Workshop_registration
1188 AFTER INSERT, UPDATE AS

```



```

BEGIN
1190     DECLARE @ReservationId int = (SELECT reservation_id FROM inserted)
      IF ((SELECT active FROM Workshop_reservations WHERE reservation_id = @ReservationId)
1192         = 0)
      BEGIN
1194         DECLARE @message varchar(100) = 'corresponding reservation is not active';
      THROW 52000, @message, 1 ROLLBACK TRANSACTION
1196     END
END

```

../Triggers/Workshops.sql

Generator danych

```
1000 from ParticipantsGenerator import *
1001 from ClientsGenerator import *
1002 from ConfDayResGenerator import *
1003 from ConfDaysGenerator import *
1004 from EsdGenerator import *
1005 from ConferenceGenerator import *
1006 from WorkshopGenerator import *
1007 from WorkshopResGen import *
1008 from ConfRegistrationGen import *
1009 from WorkshopRegistrationGen import *
1010 from PaymentGen import *

1011
1012 from random import Random
1013 from faker import Faker
1014
1015 faker = Faker(['pl_PL'])
1016 rand = Random()
1017
1018 generators = []
1019
1020 part_gen = ParticipantsGenerator(rand)
1021 part_gen.make(None, 4000)
1022 #part_gen.make(None, 5)
1023
1024 clients_gen = ClientsGenerator(part_gen)
1025 generators.append(clients_gen)
1026 clients_gen.make(400)
1027 #clients_gen.make(10)
1028
1029 conf_gen = ConferenceGenerator(rand, faker)
1030 generators.append(conf_gen)
1031 conf_gen.make(72)
1032 #conf_gen.make(3)
1033
1034 day_gen = ConfDaysGenerator(rand, faker)
1035 generators.append(day_gen)
1036 day_gen.make(conf_gen.conferences)
1037
1038 esd_gen = EsdGenerator(rand, faker)
1039 generators.append(esd_gen)
1040 esd_gen.make(day_gen.days)
1041
1042 day_res_gen = ConfDayResGenerator(clients_gen, rand, faker)
1043 generators.append(day_res_gen)
1044 day_res_gen.make(day_gen.days)
1045
1046 wor_gen = WorkshopGenerator(part_gen, rand, faker)
1047 generators.append(wor_gen)
1048 wor_gen.make(day_gen.days)
1049
1050 wor_res_gen = WorkshopResGen(day_res_gen)
1051 generators.append(wor_res_gen)
1052 wor_res_gen.make(wor_gen.workshops)
1053
1054 conf_reg_gen = ConfRegistrationGen(rand, part_gen)
1055 generators.append(conf_reg_gen)
1056 conf_reg_gen.make(day_res_gen.reservations)
1057
1058 wor_reg_gen = WorkshopRegistrationGen(rand, part_gen)
1059 generators.append(wor_reg_gen)
1060 wor_reg_gen.make(wor_res_gen.reservations)
1061
1062 pay_gen = PaymentGen(rand, faker)
1063 generators.append(pay_gen)
1064 pay_gen.make(day_res_gen.reservations)
1065
1066 for g in generators:
1067     print(g.to_sql())
```

../Generator/main.py

```
1000 from datetime import datetime, timedelta, time
```

```

1002 class AbstractClass:
1004     def random_time(self, date):
1006         return datetime.combine(date, time(self.rand.randint(0, 23), self.rand.randint(0, 59))
1008     )

```

../Generator/AbstractClass.py

```

1000 def table_to_sql(table, n_row=True):
1002     res = '\n' if n_row else ''
1004     res += table[0].to_sql()
1006     lid = 0
1008     for v in range(1, len(table)):
1010         if v - lid < 950:
1012             res += ','
1014             res += table[v].to_sql(False)
1016         else:
1018             res += '\n'
1020             res += table[v].to_sql()
1022             lid = v
1024     return res

```

../Generator/AbstractGenerator.py

```

1000 class Client:
1002     def __init__(self, cl_id, faker):
1004         self.cl_id = cl_id
1006         self.faker = faker
1008
1010         add = self.faker.address().split('\n')
1012         self.address = add[0]
1014         self.zip_code = add[-1].split(' ')[0]
1016         self.city = ' '.join(add[-1].split(' ')[1:])
1018
1020     def to_sql(self, start=True):
1022         values = "(" + str(self.cl_id) + ",\'" + self.zip_code + "\',\'" + self.city + "\',\'"
1024         + self.address + "\')'"
1026         return "INSERT INTO CLIENTS (id,zip_code, city, address) " \
1028             + "VALUES " + values if start else values

```

../Generator/Client.py

```

1000 from faker import Faker
1002 import random
1004 from ParticipantsGenerator import *
1006 from Client import *
1008 from Company import *
1010
1012 def table_to_sql(table):
1014     res = '\n'
1016     res += table[0].to_sql()
1018     lid = 0
1020     for v in range(1, len(table)):
1022         if v - lid < 950:
1024             res += ','
1026             res += table[v].to_sql(False)
1028         else:
1030             lid = v
1032             res += '\n'
1034             res += table[v].to_sql()
1036     return res
1038
1040 class ClientsGenerator:
1042     def __init__(self, participants_gen, next_client_id=1):
1044         self.faker = Faker(['pl.PL'])
1046         self.rand = random.Random()
1048
1050         self.next_client_id = next_client_id
1052         self.participants_gen = participants_gen
1054         self.clients = []
1056         self.companies = []
1058
1060     def choice(self):

```

```

1034         return self.rand.choice(self.clients)
1036
1038     def clients_count(self):
1039         return len(self.clients)
1040
1042     def make(self, n=1):
1043         for _ in range(n):
1044             cl = Client(self.next_client_id, self.faker)
1045             if self.rand.randint(0, 1) == 0:
1046                 cm = Company(self.next_client_id, self.faker, self.rand)
1047                 self.companies.append(cm)
1048             else:
1049                 self.participants_gen.make(self.next_client_id)
1050                 self.next_client_id += 1
1051                 self.clients.append(cl)
1052
1054     def to_sql(self):
1055         res = 'SET IDENTITY_INSERT Clients ON'
1056         res += table_to_sql(self.clients)
1057
1058         res += '\nSET IDENTITY_INSERT Clients OFF'
1059         res += table_to_sql(self.companies)
1060         res += '\n'
1061         res += self.participants_gen.to_sql()
1062
1063         self.clients = []
1064         self.companies = []
1065         return res

```

../Generator/ClientsGenerator.py

```

1000 class Company:
1001     def __init__(self, clients_id, faker, rand):
1002         self.faker = faker
1003         self.rand = rand
1004         self.clients_id = clients_id
1005
1006         self.name = self.faker.company()
1007         self.phone = self.faker.phone_number()
1008         self.email = self.faker.email()
1009         self.nip = self.random_nip()
1010
1012     def random_nip(self):
1013         res = ''
1014         sum = 0
1015         weights = [6, 5, 7, 2, 3, 4, 5, 6, 7]
1016         for i in range(8):
1017             k = self.rand.randint(1 if i < 3 else 0, 9)
1018             sum += weights[i] * k
1019             res += str(k)
1020         k = self.rand.randint(0, 9)
1021         if (sum + (k * weights[8])) % 11 == 10:
1022             k += (1 if k + 1 < 10 else -1)
1023         res += str(k)
1024         sum += k * weights[8]
1025         res += str(sum % 11)
1026         return res
1027
1028     def to_sql(self, start=True):
1029         values = "(" + self.name + ", " + self.nip + ", " + self.phone + ", " + str(
1030             self.clients_id) + ", " + self.email + ")"
1031         return "INSERT INTO COMPANIES (companyName, nip, phone, clients_id, email) VALUES "+
1032             values if start else values

```

../Generator/Company.py

```

1000 from AbstractClass import *
1001
1002 class ConfDayReservation(AbstractClass):
1003     def __init__(self, res_id, clients_id, day, part_count, faker, rand):
1004         self.faker = faker
1005         self.rand = rand
1006
1007         self.res_id = res_id
1008         self.day_id = day.day_id

```

```

1010         self.clients_id = clients_id

1012         self.date = self.random_time(self.faker.date.between(start_date=datetime.today(),
end_date=day.date))
        self.active = 1 if self.rand.randint(1, 1000) % 8 == 0 else 0
1014         self.due_price = self.date + timedelta(weeks=self.rand.randint(1, 4))
        self.adult_seats = self.rand.randint(1, min(day.free_seats, part_count))
1016         self.student_seats = self.rand.randint(0, max(0, min(day.free_seats, part_count) -
self.adult_seats))

1018         self.workshops_price = 0
        self.day_price = day.price * self.adult_seats
1020         self.day_price += day.price * (1 - day.stud_disc) * self.student_seats
        esds = list(filter(lambda x: x.date > self.date.date(), day.esds))
1022         esds = sorted(esds, key=lambda x: x.date)
        esd = 0 if len(esds) == 0 else esds[0].discount
1024         self.day_price *= (1 - esd)

1026     def to_sql(self, start=True):
        values = "(" + str(
1028             self.res_id) + "," + str(self.day_id) + "," + str(self.clients_id) + "," + str(
self.date) + "," + str(
            self.active) + "," + str(self.due_price) + "," + str(self.adult_seats) + ","
+ str(
1030             self.student_seats) + ")"
        return "INSERT INTO Conference_day_reservations (reservation_id, conference_day_id,
clients_id, reservation_date, active, due_price, adult_seats, student_seats) VALUES " +
values if start else values

```

../Generator/ConfDayReservation.py

```

1000 from random import Random
1001 from faker import Faker
1002 from ConfDayReservation import *

1004 class ConfDayResGenerator:
1006     def __init__(self, clients_gen, rand=Random(), faker=Faker(['pl-PL']), next_res_id=1):
        self.faker = faker
1008         self.rand = rand

1010         self.clients_gen = clients_gen
        self.next_res_id = next_res_id
1012         self.reservations = []

1014     def choice(self):
        return self.rand.choice(self.reservations)

1016     def res_count(self):
1018         return len(self.reservations)

1020     def to_sql(self):
        res = 'SET IDENTITY_INSERT Conference_day_reservations ON'
1022         res += '\n'
        res += self.reservations[0].to_sql()
1024         for v in range(1, len(self.reservations)):
            res += ','
1026             res += self.reservations[v].to_sql(False)
        res += '\nSET IDENTITY_INSERT Conference_day_reservations OFF'
1028         self.reservations = []
        return res

1030     def make(self, days):
1032         for day in days:
            n_res = self.rand.randint(2, self.clients_gen.clients_count() / 5)
1034             while day.free_seats > 0 and n_res > 0:
                n_res -= 1
1036                 self.reservations.append(
                    ConfDayReservation(self.next_res_id, self.clients_gen.choice().cl_id, day,
1038                                     len(self.clients_gen.participants_gen.participants),
self.faker, self.rand))
                self.next_res_id += 1
1040                 day.free_seats -= self.reservations[-1].adult_seats
                day.free_seats -= self.reservations[-1].student_seats

```

../Generator/ConfDayResGenerator.py

```

1000 from faker import Faker
1001 from random import Random
1002 from ConferenceDay import *
1003 from datetime import datetime, timedelta, time
1004 from AbstractGenerator import table_to_sql
1005
1006 class ConfDaysGenerator:
1007     def __init__(self, rand=Random(), faker=Faker(['pl_PL']), next_day_id=1):
1008         self.faker = faker
1009         self.rand = rand
1010
1011         self.next_day_id = next_day_id
1012         self.days = []
1013
1014     def make(self, conferences):
1015         for c in conferences:
1016             self.days.append(ConferenceDay(self.next_day_id, c.conf_id, self.faker, self.rand)
1017 )
1018             self.next_day_id += 1
1019             date = self.days[-1].date
1020             n = self.rand.randint(2, 4)
1021             for _ in range(n):
1022                 date += timedelta(days=1)
1023                 self.days.append(ConferenceDay(self.next_day_id, c.conf_id, self.faker, self.
1024 rand, date))
1025                 self.next_day_id += 1
1026
1027     def to_sql(self):
1028         res = 'SET IDENTITY_INSERT Conference_days ON'
1029         res += table_to_sql(self.days)
1030         res += '\nSET IDENTITY_INSERT Conference_days OFF'
1031         self.days = []
1032         return res

```

../Generator/ConfDaysGenerator.py

```

1000 class ConferenceDay:
1001     def __init__(self, day_id, conf_id, faker, rand, day=None):
1002         self.faker = faker
1003         self.rand = rand
1004
1005         self.day_id = day_id
1006         self.conf_id = conf_id
1007         self.price = round(self.rand.uniform(10.0, 1000.0), 2)
1008         self.stud_disc = round(self.rand.uniform(0.0, 0.5), 2)
1009         self.date = (self.faker.date_between(start_date='today', end_date='+5y')) if day is
1010 None else day
1011         self.numb_of_seats = self.rand.randint(150, 250)
1012         self.free_seats = self.numb_of_seats
1013         self.esds = []
1014
1015     def to_sql(self, start=True):
1016         values = "(" + str(self.day_id) + "," + str(self.conf_id) + ",\'" + str(self.date) + "
1017 \',\' + str(
1018             self.price) + "," + str(self.stud_disc) + ',\' + str(self.numb_of_seats) + ")"
1019         return "INSERT INTO Conference_days (conference_day_id, conference_id, date,
1020 standard_price, student_discount, number_of_seats) VALUES " + values if start else values

```

../Generator/ConferenceDay.py

```

1000 from faker import Faker
1001 from random import Random
1002 from Conference import *
1003 from AbstractGenerator import table_to_sql
1004
1005 class ConferenceGenerator:
1006     def __init__(self, rand=Random(), faker=Faker(['pl_PL']), next_conference_id=1):
1007         self.faker = faker
1008         self.rand = rand
1009
1010         self.next_conference_id = next_conference_id
1011         self.conferences = []

```

```

1014     def to_sql(self):
1015         res = 'SET IDENTITY_INSERT Conferences ON'
1016         res += table_to_sql(self.conferences)
1017         res += '\nSET IDENTITY_INSERT Conferences OFF'
1018         self.conferences = []
1019         return res
1020
1021     def make(self, n=1):
1022         for _ in range(n):
1023             self.conferences.append(Conference(self.next_conference_id, self.faker))
1024             self.next_conference_id += 1

```

../Generator/ConferenceGenerator.py

```

1000 class Conference:
1001     def __init__(self, conf_id, faker):
1002         self.faker = faker
1003
1004         self.conf_id = conf_id
1005         self.name = self.faker.bs()
1006         self.description = self.faker.text()
1007
1008     def to_sql(self, start=True):
1009         values = "(" + str(self.conf_id) + ",\'" + self.name + '\',\'" + self.description + "
1010         \'")"
1011         return "INSERT INTO Conferences (Conference_id, name, description) VALUES " + values
1012     if start else values

```

../Generator/Conference.py

```

1000 from ConfRegistration import *
1001 from AbstractGenerator import *
1002
1003 class ConfRegistrationGen:
1004     def __init__(self, rand, part_gen):
1005         self.rand = rand
1006
1007         self.part_gen = part_gen
1008         self.registrations = []
1009
1010     def to_sql(self):
1011         res = table_to_sql(self.registrations, False)
1012         self.registrations = []
1013         return res
1014
1015     def make(self, reservations):
1016         for res in reservations:
1017             if res.active == 0:
1018                 continue
1019             parts = set([p.part_id for p in self.part_gen.participants])
1020
1021             for _ in range(res.adult_seats):
1022                 p = self.rand.sample(parts, 1)
1023                 self.registrations.append(ConfRegistration(res.res_id, p[0], 1, self.rand))
1024                 parts.remove(p[0])
1025
1026             for _ in range(res.student_seats):
1027                 p = self.rand.sample(parts, 1)
1028                 self.registrations.append(ConfRegistration(res.res_id, p[0], 0, self.rand))
1029                 parts.remove(p[0])
1030

```

../Generator/ConfRegistrationGen.py

```

1000 class ConfRegistration:
1001     def __init__(self, res_id, participant_id, student, rand):
1002         self.rand = rand
1003
1004         self.res_id = res_id
1005         self.student = student
1006         self.participant_id = participant_id
1007
1008     def to_sql(self, start=True):
1009         values = "(" + str(self.res_id) + ",," + str(self.participant_id) + ",," + str(self.
1010         student) + ")"

```

```

1010         return "INSERT INTO Conference_day_registration (reservation_id , Participant_id ,
is_student) VALUES " + values if start else values

```

../Generator/ConfRegistration.py

```

1000 from random import Random
1001 from faker import Faker
1002 from Esd import *
1003 from AbstractGenerator import *
1004
1005 class EsdGenerator:
1006     def __init__(self, rand=Random(), faker=Faker(['pl_PL'])):
1007         self.faker = faker
1008         self.rand = rand
1009
1010         self.esds = []
1011
1012     def to_sql(self):
1013         res = table_to_sql(self.esds, False)
1014         self.esds = []
1015         return res
1016
1017     def make(self, days):
1018         for day in days:
1019             for _ in range(self.rand.randint(1, 5)):
1020                 e = Esd(self.faker, self.rand, day)
1021                 self.esds.append(e)
1022                 day.esds.append(e)

```

../Generator/EsdGenerator.py

```

1000 class Esd:
1001     def __init__(self, faker, rand, day):
1002         self.faker = faker
1003         self.rand = rand
1004
1005         self.day_id = day.day_id
1006         self.discount = round(self.rand.uniform(0.0, 0.5), 2)
1007         self.date = self.faker.date_between(start_date='-2y', end_date=day.date)
1008
1009     def to_sql(self, start=True):
1010         values = "(" + str(self.day_id) + ",\'" + str(self.date) + "\', " + str(self.discount)
1011         + ")"
1012         return "INSERT INTO Early_signup_discounts (conference_day_id, end_date, discount)
VALUES " + values if start else values

```

../Generator/Esd.py

```

1000 class Participant:
1001     def __init__(self, part_id, faker, client_id=None):
1002         self.part_id = part_id
1003         self.client_id = client_id
1004         self.faker = faker
1005
1006         n = self.faker.name().split(' ')
1007         if n[0] == 'pani' or n[0] == 'pan' or n[0] == 'Pan' or n[0] == 'Pani':
1008             n = n[1:]
1009         self.name = n[0]
1010         self.surname = ' '.join(n[1:])
1011         self.phone = self.faker.phone_number()
1012         self.email = self.faker.email()
1013
1014     def to_sql(self, start=True):
1015         values = "(" + str(self.part_id) + ", " + (str(
1016             self.client_id if self.client_id is not None else 'null') + ",\'" + self.name + "
\'',\'" + self.surname + "\',\'" + self.email + "\',\'" + self.phone + "\')")
1017         return "INSERT INTO Participants (participant_id, clients_id, name, surname, email,
phone) VALUES " + values if start else values

```

../Generator/Participant.py

```

1000 from faker import Faker
1001 from Participant import *
1002 from AbstractGenerator import table_to_sql

```



```

1004 class ParticipantsGenerator:
1006     def __init__(self, rand, faker=Faker(['pl.PL']), next_participant_id=1):
1008         self.faker = faker
1009         self.rand = rand
1010
1011         self.next_participant_id = next_participant_id
1012         self.participants = []
1013
1014     def choice(self):
1015         return self.rand.choice(self.participants)
1016
1017     def part_count(self):
1018         return len(self.participants)
1019
1020     def make(self, clients_id=None, n=1):
1021         for _ in range(n):
1022             res = Participant(self.next_participant_id, self.faker, clients_id)
1023             self.next_participant_id += 1
1024             self.participants.append(res)
1025
1026     def to_sql(self):
1027         res = 'SET IDENTITY_INSERT Participants ON'
1028         res += table_to_sql(self.participants)
1029         res += '\nSET IDENTITY_INSERT Participants OFF'
1030         self.participants = []
1031         return res

```

../Generator/ParticipantsGenerator.py

```

1000 from Payment import *
1001 from AbstractGenerator import *
1002
1003
1004 class PaymentGen:
1005     def __init__(self, rand, faker):
1006         self.rand = rand
1007         self.faker = faker
1008         self.payments = []
1009
1010     def make(self, reservations):
1011         for res in reservations:
1012             if res.active == 0 and self.rand.randint(1,1000) % 8 != 0:
1013                 continue
1014             price = res.workshops_price + res.day_price
1015             payed = int(price)
1016             if self.rand.randint(1, 1000) % 30 == 0: # nie opłacone
1017                 payed = 0 if self.rand.randint(0, 1000) % 2 == 0 else self.rand.randint(0, int
1018 (price))
1019             if payed == 0:
1020                 continue
1021             n_payments = self.rand.randint(1, 4)
1022             values = [payed // n_payments for _ in range(n_payments)]
1023             i = 0
1024             while sum(values) < payed:
1025                 values[i % n_payments] += 1
1026                 i += 1
1027             i = 0
1028             while i < n_payments:
1029                 p = self.rand.randint(0, values[i])
1030                 values[i] -= p
1031                 values[(i + 1) % n_payments] += p
1032                 i += 1
1033             values[-1] += round(price - int(price), 2)
1034             for value in values:
1035                 self.payments.append(Payment(value, res, self.faker, self.rand))
1036
1037     def to_sql(self):
1038         res = table_to_sql(self.payments, False)
1039         self.payments = []
1040         return res

```

../Generator/PaymentGen.py

```

1000 from AbstractClass import *

```

```

1002 class Payment(AbstractClass):
1004     def __init__(self, value, day_res, faker, rand):
1006         self.faker = faker
1007         self.rand = rand
1008
1009         self.res_id = day_res.res_id
1010         self.date = self.random_time(self.faker.date_between(start_date=day_res.date, end_date
1011 =day_res.due_price))
1012         self.value = value
1013
1014     def to_sql(self, start=True):
1015         values = "(" + str(self.res_id) + ",\'" + str(self.date) + "\'," + str(self.value) + "
1016 )"
1017         return "INSERT INTO Payments (reservation_id, in_date, value) VALUES " + values if
1018 start else values

```

../Generator/Payment.py

```

1000 from random import Random
1001 from faker import Faker
1002 from Workshop import *
1003 from AbstractGenerator import table_to_sql
1004
1005 class WorkshopGenerator:
1006     def __init__(self, part_gen, rand=Random(), faker=Faker(['pl_PL']), next_workshop_id=1):
1007         self.faker = faker
1008         self.rand = rand
1009
1010         self.part_gen = part_gen
1011         self.workshops = []
1012         self.next_workshop_id = next_workshop_id
1013
1014     def to_sql(self):
1015         res = 'SET IDENTITY_INSERT Workshops ON'
1016         res += table_to_sql(self.workshops)
1017         res += '\nSET IDENTITY_INSERT Workshops OFF'
1018         self.workshops = []
1019         return res
1020
1021     def make(self, days):
1022         for day in days:
1023             for _ in range(self.rand.randint(2, 6)):
1024                 self.workshops.append(
1025                     Workshop(self.next_workshop_id, day, self.part_gen.part_count(), self.
1026 faker, self.rand))
1027                 self.next_workshop_id += 1

```

../Generator/WorkshopGenerator.py

```

1000 from datetime import datetime, timedelta, time
1001
1002 class Workshop:
1003     def __init__(self, workshop_id, day, part_count, faker, rand):
1004         self.faker = faker
1005         self.rand = rand
1006
1007         self.workshop_id = workshop_id
1008         self.day_id = day.day_id
1009         self.price = round(self.rand.uniform(10.0, 1000.0), 2)
1010         self.start = self.random_time(day.date)
1011         self.end = self.random_time(day.date)
1012         if self.start > self.end:
1013             self.start, self.end = self.end, self.start
1014         self.topic = self.faker.bs()
1015         self.numb_seats = self.rand.randint(1, min(day.numb_of_seats, part_count))
1016         self.free_seats = self.numb_seats
1017
1018     def random_time(self, date):
1019         return datetime.combine(date, time(self.rand.randint(0, 23), self.rand.randint(0, 59))
1020 )
1021
1022     def to_sql(self, start=True):

```

```

        values = "(" + str(self.workshop_id) + "," + str(self.day_id) + ",\'" + str(self.start
    ) + "\',\'" + str(
1024         self.end) + "\',\'" + self.topic + "\',\'" + str(self.price) + "," + str(self.
        numb_seats) + ")"
        return "INSERT INTO Workshops (workshop_id, conference_day_id, start_time, end_time,
        topic, price, number_of_seats) VALUES " + values if start else values

```

../Generator/Workshop.py

```

1000 from WorkshopRegistration import *
1002 from AbstractGenerator import *

1004 class WorkshopRegistrationGen:
1006     def __init__(self, rand, part_gen):
1008         self.rand = rand
1010         self.part_gen = part_gen
1012         self.registrations = []

1014     def to_sql(self):
1016         res = table_to_sql(self.registrations, False)
1018         self.registrations = []
1020         return res

1022     def make(self, reservations):
1024         for res in reservations:
            if res.active == 0:
                continue
            parts = set([p.part_id for p in self.part_gen.participants])
            for _ in range(res.nr_seats):
                p = self.rand.sample(parts, 1)
                self.registrations.append(WorkshopRegistration(res.res_id, p[0]))
                parts.remove(p[0])

```

../Generator/WorkshopRegistrationGen.py

```

1000 class WorkshopRegistration:
1002     def __init__(self, res_id, participant_id):
1004         self.res_id = res_id
1006         self.participant_id = participant_id

1008     def to_sql(self, start=True):
1010         values = "(" + str(self.res_id) + "," + str(self.participant_id) + ")"
1012         return "INSERT INTO Workshop-registration (reservation_id, Participant_id) VALUES " +
1014         values if start else values

```

../Generator/WorkshopRegistration.py

```

1000 from random import Random
1002 from faker import Faker
1004 from WorkshopRes import *
1006 from AbstractGenerator import table_to_sql

1008 class WorkshopResGen:
1010     def __init__(self, conf_day_res_gen, rand=Random(), faker=Faker(['pl.PL']), next_res_id=1)
1012         :
1014         self.faker = faker
1016         self.rand = rand

1018         self.conf_day_res_gen = conf_day_res_gen
1020         self.next_res_id = next_res_id
1022         self.reservations = []

1024     def to_sql(self):
        res = 'SET IDENTITY_INSERT Workshop-reservations ON'
        res += table_to_sql(self.reservations)
        res += '\nSET IDENTITY_INSERT Workshop-reservations OFF'
        self.reservations = []
        return res

    def make(self, workshops):
        for work in workshops:
            n_res = self.rand.randint(1, self.conf_day_res_gen.res_count() // 4)

```

```

1026         while work.free_seats > 0 and n_res > 0:
1027             n_res -= 1
1028             self.reservations.append(
1029                 WorkshopRes(self.next_res_id, work, self.conf_day_res_gen.choice(), self.
1030 faker, self.rand))
1031             self.next_res_id += 1
1032             work.free_seats -= self.reservations[-1].nr_seats

```

../Generator/WorkshopResGen.py

```

1000 from AbstractClass import *
1001
1002 class WorkshopRes(AbstractClass):
1003     def __init__(self, res_id, workshop, conf_res, faker, rand):
1004         self.faker = faker
1005         self.rand = rand
1006
1007         self.res_id = res_id
1008         self.work_id = workshop.workshop_id
1009         self.date = self.random_time(
1010             self.faker.date_between(start_date=datetime.today(), end_date=workshop.start.date
1011 ()))
1012         self.due_price = self.date + timedelta(weeks=self.rand.randint(1, 4))
1013         self.nr_seats = self.rand.randint(1, workshop.free_seats)
1014         self.conf_res_id = conf_res.res_id
1015         self.active = self.rand.randint(0, 1)
1016
1017         conf_res.workshops_price += self.nr_seats * workshop.price
1018
1019     def to_sql(self, start=True):
1020         values = "(" + str(
1021             self.res_id) + "," + str(self.work_id) + ",\'" + str(self.date) + "\',\'" + str(
1022             self.due_price) + "\',\'" + str(
1023             self.nr_seats) + ',\' + str(self.conf_res_id) + ',\' + str(self.active) + ")"
1024         return "INSERT INTO Workshop_reservations (reservation_id, workshop_id,
1025 reservation_date, due_price, nr_of_seats, Conference_day_res_id, active) VALUES " + values
1026         if start else values

```

../Generator/WorkshopRes.py

Funkcje realizowane przez system

Funkcje użytkowników:

- Administrator - Dodawanie pracowników
- Pracownik – Dodawanie organizatorów
- Organizator – Utworzenie nowej konferencji
- Organizator – Wprowadzenie informacji o kolejnych dniach konferencji
- Organizator - Wprowadzenie informacji o warsztatach
- Klient - Rejestracja w charakterze uczestnika / firmy w systemie
- Klient - Rezerwacja miejsc na konferencji
- Klient – Rezerwacja miejsc na warsztatach
- Uczestnik - Wprowadzenie danych osobowych
- Organizator - Generowanie raportów o brakujących danych osobowych
- Organizator - Generowanie identyfikatorów
- Organizator - Generowanie raportów o zarejestrowanych uczestnikach na każdy dzień i warsztat
- Organizator – Wprowadzanie progów cenowych
- Organizator – Wprowadzanie cen warsztatów
- Organizator - Generowanie informacji o płatnościach
- Organizator – Wprowadzenie informacji o wpływającej wpłacie od Klienta
- Pracownik - Generowanie danych statystycznych na temat klientów

Funkcje systemowe:

- Zapewnienie, by wszyscy zapisujący się na warsztaty spełniali wymogi (np. by byli zarejestrowani na konferencję)
- Obsługa zapisów na warsztaty (limit miejsc, inne warsztaty)
- Kalkulacja sumarycznego kosztu ponoszonego przez uczestnika w ramach udziału w konferencji
- Kalkulacja sumarycznego kosztu ponoszonego przez klienta
- Informacja o dostępnych wolnych miejscach na konferencję
- Rejestracja płatności klientów, obsługa nieopłaconych rezerwacji
- Tworzenie raportów dla organizatorów – najbardziej aktywni klienci, informacje o płatnościach klientów

Uprawnienia

1. Administrator - Całkowity dostęp do bazy.
2. Pracownik – Dostęp poziomu organizatora dla wszystkich konferencji.
3. Organizator – Dostęp do wszystkich rekordów powiązanych z konkretną konferencją. Możliwość edycji tabel warsztatów, dni konferencyjnych.
4. Klient – Rejestracja na konferencje i warsztaty, wprowadzanie danych osobowych.
5. Uczestnik – Wgląd i edycja wprowadzonych danych osobowych, podgląd warsztatów i dni konferencyjnych na jakie jest zapisany (nie może generować kosztów, jako że jest podpięty pod klienta).