PODSTAWY BAZ DANYCH - PROJEKT
System bazodanowy dla firmy oferującej kursy i szkolenia

Jakub Kaliński | Michał Szymocha | Emil Żychowicz
Informatyka | rok II

Wstęp

Firma zajmująca się organizowaniem kursów i szkoleń, zarówno stacjonarnych, jak i zdalnych, zleciła stworzenie systemu bazodanowego, który ma na celu efektywne zarządzanie różnorodnymi usługami edukacyjnymi. W odpowiedzi na zmiany rynkowe, w tym wpływ pandemii COVID-19, oferta została dostosowana do modelu hybrydowego, w którym uczestnicy mogą brać udział zarówno w wydarzeniach na żywo, jak i korzystać z nagrań lub materiałów dostępnych online.

System bazodanowy będzie odpowiadał za obsługę takich usług jak webinary, kursy oraz studia, które różnią się zarówno formą (stacjonarne, online, hybrydowe), jak i wymaganiami dotyczącymi frekwencji oraz zaliczenia. System będzie musiał przechowywać dane o uczestnikach, wykładowcach, kursach, modułach oraz planach zajęć, umożliwiając jednocześnie integrację z zewnętrznymi systemami płatności. Dodatkowo, system zapewni generowanie raportów dotyczących frekwencji, płatności i innych informacji istotnych z perspektywy zarządzania działalnością edukacyjną.

W odpowiedzi na rosnące zapotrzebowanie na elastyczne formy kształcenia, system bazodanowy ma na celu ułatwienie procesu zarządzania szkoleniami, zapewniając wygodny dostęp do danych oraz narzędzi do monitorowania postępów uczestników.

Spis treści

1 Funkcje i użytkownicy systemu . . . 9
1.1 Model uprawnień w systemie . . . 9
1.2 Użytkownicy systemu (z odwzorowaniem hierarchii) . . . 9
1.2.1 Schemat użytkowników systemu . . . 9
1.3 Funkcje systemu . . . 10
1.3.1 Konta i uprawnienia . . . 10
1.3.2 Webinary . . . 10
1.3.3 Kursy . . . 10
1.3.4 Studia . . . 11
1.3.5 Koszyk – system płatności . . . 12
1.3.6 Raporty . . . 12
1.3.7 Dodatkowe funkcje systemu . . . 12
2 Schemat bazy danych . . . 13
3 Generowanie danych testowych do bazy SQL . . . 13
3.1 Biblioteki i konfiguracje . . . 13
3.2 Generowane dane . . . 13
3.3 Struktura i logika . . . 13
3.4 Generowanie danych do bazy SQL . . . 14
3.5 Specjalne funkcje . . . 14
4 Opisy tabel . . . 14
4.1 Sekcja studiów . . . 14
4.1.1 Tabela AsyncClassDetails . . . 14
4.1.2 Tabela Atonements . . . 15
4.1.3 Tabela ClassMeeting . . . 15

4.1.4 Tabela Convention . . . 16
4.1.5 Tabela Grades . . . 17
4.1.6 Tabela Internship . . . 17
4.1.7 Tabela InternshipDetails . . . 18
4.1.8 Tabela OfflineVideoClass . . . 18
4.1.9 Tabela OnlineLiveClass . . . 19
4.1.10 Tabela SemesterDetails . . . 20
4.1.11 Tabela StationaryClass . . . 20
4.1.12 Tabela Studies . . . 21
4.1.13 Tabela StudiesDetails . . . 22
4.1.14 Tabela Subject . . . 23
4.1.15 Tabela SubjectDetails . . . 24
4.1.16 Tabela SubjectStudiesBridge . . . 24
4.1.17 Tabela SyncClassDetails . . . 24
4.2 Sekcja webinarów . . . 25
4.2.1 Tabela WebinarDetails . . . 25
4.2.2 Tabela Webinars . . . 26
4.3 Sekcja kursów . . . 27
4.3.1 Tabela Courses . . . 27
4.3.2 Tabela Modules . . . 28
4.3.3 Tabela StationaryMeeting . . . 29
4.3.4 Tabela StationaryMeetingDetails . . . 30
4.3.5 Tabela OfflineVideo . . . 30
4.3.6 Tabela OfflineVideoDetails . . . 31
4.3.7 Tabela OnlineLiveMeeting . . . 32
4.3.8 Tabela OnlineLiveMeetingDetails . . . 32
4.3.9 Tabela CourseParticipants . . . 33
4.4 Sekcja systemu opłat . . . 34
4.4.1 Tabela Payments . . . 34
4.4.2 Tabela OrderDetails . . . 34
4.4.3 Tabela Orders . . . 35
4.4.4 Tabela Services . . . 35
4.4.5 Tabela ClassMeetingService . . . 37
4.4.6 Tabela StudiesService . . . 37
4.4.7 Tabela ConventionService . . . 38
4.4.8 Tabela CourseService . . . 38
4.4.9 Tabela WebinarService . . . 38
4.5 Sekcja pomieszczeń . . . 39
4.5.1 Tabela Rooms . . . 39
4.5.2 Tabela RoomDetails . . . 40
4.5.3 Tabela RoomSchedule . . . 40
4.6 Sekcja pracowników . . . 41
4.6.1 Tabela Degrees . . . 41
4.6.2 Tabela EmployeeDegree . . . 41
4.6.3 Tabela Employees . . . 42
4.6.4 Tabela EmployeesSuperior . . . 43
4.6.5 Tabela Languages . . . 43
4.6.6 Tabela Translators . . . 44
4.6.7 Tabela TranslatorsLanguages . . . 45
4.7 Sekcja użytkowników . . . 45
4.7.1 Tabela Locations . . . 45
4.7.2 Tabela ServiceUserDetails . . . 46
4.7.3 Tabela UserAddressDetails . . . 47
4.7.4 Tabela UserContact . . . 48
4.7.5 Tabela UserType . . . 48
4.7.6 Tabela UserTypePermissionsHierarchy . . . 49
4.7.7 Tabela Users . . . 49
5 Widoki . . . 50
5.0.1 Widok ATTENDANCE_LISTS_COURSES - Emil Żychowicz . . . 50
5.0.2 Widok ATTENDANCE_LISTS_OFFLINEVIDEO_COURSES - Emil Żychowicz . . . 51

5.0.3 Widok ATTENDANCE_MEETINGS_IN_COURSES - Emil Żychowicz . . . 52
5.0.4 Widok CLASSMEETING_USERS - Emil Żychowicz . . . 52
5.0.5 Widok CONVENTION_INCOME - Emil Żychowicz . . . 53
5.0.6 Widok CONVENTION_USERS - Emil Żychowicz . . . 53
5.0.7 Widok COURSES_USERS - Jakub Kaliński . . . 54
5.0.8 Widok COURSE_INCOME - Emil Żychowicz . . . 54
5.0.9 Widok FINANCIAL_REPORT - Emil Żychowicz . . . 54
5.0.10 Widok FULL_PRICE - Emil Żychowicz . . . 55
5.0.11 Widok IN_DEBT_USERS - Jakub Kaliński . . . 55
5.0.12 Widok PARTICIPANTS_MEETINGS_FUTURE_COURSES - Emil Żychowicz . . . 56
5.0.13 Widok NUM_OF_PARTICIPANTS_FUTURE_COURSES - Emil Żychowicz . . . 57
5.0.14 Widok NUM_OF_PARTICIPANTS_MEETINGS_COURSES - Emil Żychowicz . . . 57
5.0.15 Widok SERVICE_ID_INCOME - Emil Żychowicz . . . 58
5.0.16 Widok SERVICE_USERS - Jakub Kaliński . . . 58
5.0.17 Widok START_END_OF_CLASSMEETING - Emil Żychowicz . . . 59
5.0.18 Widok START_END_OF_CONVENTION - Emil Żychowicz . . . 59
5.0.19 Widok START_END_OF_COURSES - Emil Żychowicz . . . 59
5.0.20 Widok START_END_OF_SERVICES - Emil Żychowicz . . . 60
5.0.21 Widok START_END_STUDIES - Emil Żychowicz . . . 60
5.0.22 Widok START_END_OF_WEBINAR - Emil Żychowicz . . . 60
5.0.23 Widok STUDIES_INCOME - Emil Żychowicz . . . 61
5.0.24 Widok STUDIES_USERS - Jakub Kaliński . . . 61
5.0.25 Widok WEBINAR_USERS - Jakub Kaliński . . . 61
5.0.26 Widok WEBINAR_INCOME - Emil Żychowicz . . . 62
5.0.27 Widok COURSE_SCHEDULE - Emil Żychowicz . . . 62
5.0.28 Widok COURSE_PASSING_STATUS - Emil Żychowicz . . . 63
5.0.29 Widok COURSE_INFO - Emil Żychowicz . . . 65
5.0.30 Widok CONSUMER_BASKET - Emil Żychowicz . . . 66
5.0.31 Widok CLASS_MEETINGS_INCOME - Emil Żychowicz . . . 66
5.0.32 Widok ATTENDANCE_RAPORT - Jakub Kaliński . . . 66
5.0.33 Widok V_ConventionUsers - Michał Szymocha . . . 67
5.0.34 Widok V_StudentGrades - Michał Szymocha . . . 68
5.0.35 Widok V_SemesterSubjectsConventions - Michał Szymocha . . . 69
5.0.36 Widok V_StudiesEnrollment - Michał Szymocha . . . 69
5.0.37 Widok V_EmployeeHierarchy - Jakub Kaliński . . . 70
5.0.38 Widok V_ConventionSchedule - Michał Szymocha . . . 70
5.0.39 Widok V_SubjectMeetingSchedule - Michał Szymocha . . . 71
5.0.40 Widok V_TranslatorLanguageSkill - Jakub Kaliński . . . 72
5.0.41 Widok V_ConventionStudents - Michał Szymocha . . . 72
5.0.42 Widok V_ClassMeetingStudents - Michał Szymocha . . . 73
5.0.43 Widok AllEvents - Michał Szymocha . . . 73
5.0.44 Widok V_StudentCollidingEvents - Michał Szymocha . . . 74
5.0.45 Widok V_StudentsFinishedStudies - Michał Szymocha . . . 75
5.0.46 Widok V_WebinarsWithAttendance - Michał Szymocha . . . 76
5.0.47 Widok V_StudentSchedule - Jakub Kaliński . . . 76
5.0.48 Widok V_EmployeeSchedule - Jakub Kaliński . . . 77
5.0.49 Widok V_ClassAttendanceList - Jakub Kaliński . . . 79
5.0.50 Widok V_ClassAttendanceAggregate - Michał Szymocha . . . 79
5.0.51 Widok V_EmployeeWorkload - Michał Szymocha . . . 80
5.0.52 Widok V_StudiesInfo - Michał Szymocha . . . 80
5.0.53 Widok V_FutureStudentSchedule - Michał Szymocha . . . 80
6 Procedury . . . 81
6.0.1 Procedura p_AddRoom - Michał Szymocha . . . 81
6.0.2 Procedura p_ReserveRoom - Michał Szymocha . . . 81
6.0.3 Procedura p_EditReservation - Michał Szymocha . . . 83
6.0.4 Procedura p_CreateStudies - Michał Szymocha . . . 84
6.0.5 Procedura p_AddSubject - Michał Szymocha . . . 86
6.0.6 Procedura p_AddConvention - Michał Szymocha . . . 88
6.0.7 Procedura p_EnrollStudentInStudies - Michał Szymocha . . . 89
6.0.8 Procedura p_CreateStationaryClass - Michał Szymocha . . . 91

6.0.9 Procedura p_CreateOnlineLiveClass - Michał Szymocha . . . 94
6.0.10 Procedura p_CreateOfflineVideoClass - Michał Szymocha . . . 97
6.0.11 Procedura p_ChangeSubjectCoordinator - Michał Szymocha . . . 100
6.0.12 Procedura p_ChangeStudiesCoordinator - Michał Szymocha . . . 100
6.0.13 Procedura p_EditStudies - Michał Szymocha . . . 101
6.0.14 Procedura p_EditSubject - Michał Szymocha . . . 103
6.0.15 Procedura p_ChangeSubjectCoordinator - Michał Szymocha . . . 104
6.0.16 Procedura p_AddConvention - Michał Szymocha . . . 105
6.0.17 Procedura p_AddSubjectToStudies - Michał Szymocha . . . 107
6.0.18 Procedura p_ChangeClassMeetingTeacher - Michał Szymocha . . . 108
6.0.19 Procedura p_ChangeClassMeetingTranslator - Michał Szymocha . . . 108
6.0.20 Procedura p_ChangeClassMeetingLanguage - Michał Szymocha . . . 109
6.0.21 Procedura p_EditClassMeeting - Michał Szymocha . . . 110
6.0.22 Procedura p_EditStationaryClass - Michał Szymocha . . . 112
6.0.23 Procedura p_EditOnlineLiveClass - Michał Szymocha . . . 114
6.0.24 Procedura Procedura p_EditOfflineVideoClass - Michał Szymocha . . . 115
6.0.25 Procedura p_DeleteUserClassMeetingDetails - Michał Szymocha . . . 116
6.0.26 Procedura p_DeleteClassMeetingDetails - Michał Szymocha . . . 117
6.0.27 Procedura p_DeleteClassMeeting - Michał Szymocha . . . 118
6.0.28 Procedura p_DeleteConvention - Michał Szymocha . . . 119
6.0.29 Procedura p_DeleteSubjectFromStudies - Michał Szymocha . . . 121
6.0.30 Procedura p_DeleteSubject - Michał Szymocha . . . 122
6.0.31 Procedura p_CreateInternship - Michał Szymocha . . . 123
6.0.32 Procedura p_EditInternship - Michał Szymocha . . . 124
6.0.33 Procedura p_AddStudentInternship - Michał Szymocha . . . 125
6.0.34 Procedura p_InitiateInternship - Michał Szymocha . . . 126
6.0.35 Procedura p_DeleteInternship - Michał Szymocha . . . 127
6.0.36 Procedura p_DeleteInternshipDetails - Michał Szymocha . . . 128
6.0.37 Procedura p_CreateGrade - Michał Szymocha . . . 129
6.0.38 Procedura p_EnrollStudentInConvention - Michał Szymocha . . . 130
6.0.39 Procedura p_EnrollStudentInSyncClassMeeting - Michał Szymocha . . . 132
6.0.40 Procedura p_EnrollStudentInAsyncClass - Michał Szymocha . . . 133
6.0.41 Procedura p_EnrollStudentInSubject - Michał Szymocha . . . 134
6.0.42 Procedura p_StudiesSyllabus - Michał Szymocha . . . 134
6.0.43 Procedura p_ChangeAttendanceStatus . . . 135
6.0.44 Procedura p_ChangeViewStatusAsyncClass . . . 136
6.0.45 Procedura p_StudiesSemesterSchedule . . . 137
6.0.46 Procedura p_EditGrade - Michał Szymocha . . . 138
6.0.47 Procedura p_DeleteGrade - Michał Szymocha . . . 139
6.0.48 Procedura p_EditOrder - Michał Szymocha . . . 140
6.0.49 Procedura p_AddOrder - Michał Szymocha . . . 141
6.0.50 Procedura p_AddClassMeetingService - Emil Żychowicz . . . 142
6.0.51 Procedura p_AddConventionService - Emil Żychowicz . . . 142
6.0.52 Procedura p_AddCourseParticipant - Emil Żychowicz . . . 143
6.0.53 Procedura p_AddCourseService - Emil Żychowicz . . . 145
6.0.54 Procedura p_AddOfflineVideo - Emil Żychowicz . . . 146
6.0.55 Procedura p_AddOfflineVideoDetails - Emil Żychowicz . . . 147
6.0.56 Procedura p_AddOnlineLiveMeeting - Emil Żychowicz . . . 148
6.0.57 Procedura p_AddOnlineLiveMeetingDetails - Emil Żychowicz . . . 150
6.0.58 Procedura p_CreateOrder - Emil Żychowicz . . . 151
6.0.59 Procedura p_AddToOrder - Emil Żychowicz . . . 152
6.0.60 Procedura p_AddPayment - Emil Żychowicz . . . 153
6.0.61 Procedura p_AddService - Emil Żychowicz . . . 154
6.0.62 Procedura p_AddStationaryMeeting - Emil Żychowicz . . . 155
6.0.63 Procedura p_AddStationaryMeetingDetails - Emil Żychowicz . . . 156
6.0.64 Procedura p_AddStudiesService - Emil Żychowicz . . . 157
6.0.65 Procedura p_AddWebinarService - Emil Żychowicz . . . 158
6.0.66 Procedura p_CreateCourse - Emil Żychowicz . . . 159
6.0.67 Procedura p_CreateModule - Emil Żychowicz . . . 160
6.0.68 Procedura p_CreateOfflineVideo - Emil Żychowicz . . . 162

6.0.69 Procedura p_CreateOrder - Emil Żychowicz . . . 163
6.0.70 Procedura p_CreateStationaryMeeting - Emil Żychowicz . . . 164
6.0.71 Procedura p_DeleteClassMeetingService - Emil Żychowicz . . . 165
6.0.72 Procedura p_DeleteConventionService - Emil Żychowicz . . . 166
6.0.73 Procedura p_DeleteCourse - Emil Żychowicz . . . 167
6.0.74 Procedura p_DeleteCourseParticipant - Emil Żychowicz . . . 168
6.0.75 Procedura p_DeleteCourseService - Emil Żychowicz . . . 169
6.0.76 Procedura p_DeleteModule - Emil Żychowicz . . . 169
6.0.77 Procedura p_DeleteOfflineVideo - Emil Żychowicz . . . 171
6.0.78 Procedura p_DeleteOfflineVideoDetails - Emil Żychowicz . . . 172
6.0.79 Procedura p_DeleteOnlineLiveMeeting - Emil Żychowicz . . . 172
6.0.80 Procedura p_DeleteOnlineLiveMeetingDetails - Emil Żychowicz . . . 173
6.0.81 Procedura p_DeleteOrder - Emil Żychowicz . . . 174
6.0.82 Procedura p_DeleteOrderDetails - Emil Żychowicz . . . 175
6.0.83 Procedura p_DeletePayment - Emil Żychowicz . . . 175
6.0.84 Procedura p_DeleteService - Emil Żychowicz . . . 176
6.0.85 Procedura p_DeleteStationaryMeeting - Emil Żychowicz . . . 177
6.0.86 Procedura p_DeleteStationaryMeetingDetails - Emil Żychowicz . . . 178
6.0.87 Procedura p_DeleteStudiesService - Emil Żychowicz . . . 179
6.0.88 Procedura p_DeleteWebinarService - Emil Żychowicz . . . 180
6.0.89 Procedura p_EditClassMeetingService - Emil Żychowicz . . . 180
6.0.90 Procedura p_EditConventionService - Emil Żychowicz . . . 181
6.0.91 Procedura p_EditCourses - Emil Żychowicz . . . 182
6.0.92 Procedura p_EditCourseService - Emil Żychowicz . . . 184
6.0.93 Procedura p_EditModules - Emil Żychowicz . . . 185
6.0.94 Procedura p_EditOfflineVideo - Emil Żychowicz . . . 186
6.0.95 Procedura p_EditOfflineVideoDateOfViewing - Emil Żychowicz . . . 187
6.0.96 Procedura p_EditOnlineLiveAttendance - Emil Żychowicz . . . 188
6.0.97 Procedura p_EditOnlineLiveMeeting - Emil Żychowicz . . . 189
6.0.98 Procedura p_EditPayment - Emil Żychowicz . . . 190
6.0.99 Procedura p_EditStationaryMeeting - Emil Żychowicz . . . 191
6.0.100 Procedura p_EditStationaryMeetingAttendance - Emil Żychowicz . . . 193
6.0.101 Procedura p_EditStudiesService - Emil Żychowicz . . . 193
6.0.102 Procedura p_EditWebinarService - Emil Żychowicz . . . 194
6.0.103 Procedura p_FinalizeOrder - Emil Żychowicz . . . 195
6.0.104 Procedura p_UpdatePrincipalAgreement - Emil Żychowicz . . . 196
6.0.105 Procedura p_AddWebinarUser - Emil Żychowicz . . . 197
6.0.106 Procedura p_CreateWebinar - Jakub Kaliński . . . 198
6.0.107 Procedura p_EditWebinar - Jakub Kaliński . . . 199
6.0.108 Procedura p_DeleteWebinar - Jakub Kaliński . . . 200
6.0.109 Procedura p_AddUser - Jakub Kaliński . . . 200
6.0.110 Procedura p_UpdateUser - Jakub Kaliński . . . 201
6.0.111 Procedura p_DeleteUser - Jakub Kaliński . . . 202
6.0.112 Procedura p_AddEmployee - Jakub Kaliński . . . 202
6.0.113 Procedura p_UpdateEmployee - Jakub Kaliński . . . 203
6.0.114 Procedura p_DeleteEmployee - Jakub Kaliński . . . 203
6.0.115 Procedura p_AssignSupervisor - Jakub Kaliński . . . 204
6.0.116 Procedura p_AddUserAddress - Jakub Kaliński . . . 205
6.0.117 Procedura p_UpdateUserAddress - Jakub Kaliński . . . 205
6.0.118 Procedura p_DeleteUserAddress - Jakub Kaliński . . . 206
6.0.119 Procedura p_AddUserContact - Jakub Kaliński . . . 206
6.0.120 Procedura p_UpdateUserContact - Jakub Kaliński . . . 207
6.0.121 Procedura p_DeleteUserContact - Jakub Kaliński . . . 207
6.0.122 Procedura p_AddUserType - Jakub Kaliński . . . 208
6.0.123 Procedura p_UpdateUserType - Jakub Kaliński . . . 208
6.0.124 Procedura p_DeleteUserType - Jakub Kaliński . . . 209
6.0.125 Procedura p_AddDegree - Jakub Kaliński . . . 209
6.0.126 Procedura p_UpdateDegree - Jakub Kaliński . . . 210
6.0.127 Procedura p_DeleteDegree - Jakub Kaliński . . . 210
6.0.128 Procedura p_AddEmployeeDegree - Jakub Kaliński . . . 211

6.0.129 Procedura p_UpdateEmployeeDegree - Jakub Kaliński . . . 211
6.0.130 Procedura p_DeleteEmployeeDegree - Jakub Kaliński . . . 212
6.0.131 Procedura p_AddTranslator - Jakub Kaliński . . . 213
6.0.132 Procedura p_DeleteTranslator - Jakub Kaliński . . . 213
6.0.133 Procedura p_AddLanguage - Jakub Kaliński . . . 214
6.0.134 Procedura p_UpdateLanguage - Jakub Kaliński . . . 214
6.0.135 Procedura p_DeleteLanguage - Jakub Kaliński . . . 215
6.0.136 Procedura p_AddTranslatorLanguage - Jakub Kaliński . . . 215
6.0.137 Procedura p_DeleteTranslatorLanguage - Jakub Kaliński . . . 216
6.0.138 Procedura p_AddLocation - Jakub Kaliński . . . 216
6.0.139 Procedura p_UpdateLocation - Jakub Kaliński . . . 217
6.0.140 Procedura p_DeleteLocation - Jakub Kaliński . . . 217
7 Funkcje . . . 218
7.0.1 Funkcja f_CalculateAttendancePercentageOnModule - Emil Żychowicz . . . 218
7.0.2 Funkcja f_CalculateMINRoomCapacityCourse - Emil Żychowicz . . . 219
7.0.3 Funkcja f_CalculateOrderValue - Emil Żychowicz . . . 219
7.0.4 Funkcja f_CalculatePaidOrderValue - Emil Żychowicz . . . 220
7.0.5 Funkcja f_CalculatePaidServiceValue - Emil Żychowicz . . . 220
7.0.6 Funkcja f_CheckIfCourseIsPassed - Emil Żychowicz . . . 221
7.0.7 Funkcja f_CourseSchedule - Emil Żychowicz . . . 221
7.0.8 Funkcja f_GetServiceValue - Emil Żychowicz . . . 222
7.0.9 Funkcja f_IsReadyToParticipate - Emil Żychowicz . . . 223
7.0.10 Funkcja f_CalculateAverageUserAge - Jakub Kaliński . . . 224
7.0.11 Funkcja f_CountEmployeesUnderSupervisor - Jakub Kaliński . . . 224
7.0.12 Funkcja f_HasUserAddress - Jakub Kaliński . . . 224
7.0.13 Funkcja f_CountTranslatorLanguages - Jakub Kaliński . . . 225
7.0.14 Funkcja f_CountWebinarParticipants - Jakub Kaliński . . . 225
7.0.15 Funkcja f_IsUserRegisteredForWebinar - Jakub Kaliński . . . 225
7.0.16 Funkcja f_IsUserEmployee - Jakub Kaliński . . . 226
7.0.17 Funkcja p_CalculateSubjectAttendance - Michał Szymocha . . . 226
7.0.18 Funkcja CalculateAvailableSeatsStudies - Michał Szymocha . . . 227
7.0.19 Funkcja p_CalculateStudiesAttendance - Michał Szymocha . . . 227
7.0.20 Funkcja p_CalculateInternshipCompletion - Michał Szymocha . . . 228
7.0.21 Funkcja p_CalculateAverageNumberOfPeopleInClass - Michał Szymocha . . . 229
7.0.22 Funkcja p_CalculateMINRoomCapacity - Michał Szymocha . . . 229
7.0.23 Funkcja totalTimeSpentInClass - Michał Szymocha . . . 230
7.0.24 Funkcja p_CalculateAvailableSeatsStudies - Michał Szymocha . . . 230
8 Triggery . . . 230
8.0.1 Opis triggera `trg_AddPayment` - Emil Żychowicz . . . 230
8.0.2 Opis triggera `trg_AddStudentDetails` - Jakub Kaliński . . . 233
8.0.3 Opis triggera `trg_AddStudentDetailsSubject` - Michał Szymocha . . . 234
8.0.4 Opis triggera `trg_DeleteUserFromStudies` - Michał Szymocha . . . 235
8.0.5 Opis triggera `trg_DeleteUserFromSubject` - Jakub Kaliński . . . 236
9 Role . . . 236
9.0.1 Rola: `administrator` . . . 236
9.0.2 Rola: `director` . . . 236
9.0.3 Rola: `deans_office` . . . 237
9.0.4 Rola: `coordinator_studies` . . . 237
9.0.5 Rola: `coordinator_subject_module` . . . 238
9.0.6 Rola: `lecturer` . . . 238
9.0.7 Rola: `student_participant` . . . 239
9.0.8 Rola: `guest` . . . 239
9.0.9 Rola: `system` . . . 240
9.0.10 Rola: `coordinator_practices` . . . 240
10 Indeksy . . . 240
10.1 ServiceUserDetails - Jakub Kaliński . . . 240
10.1.1 Index: `IX_ServiceUserDetails_DateOfRegistration` . . . 240
10.2 Users - Jakub Kaliński . . . 240
10.2.1 Index: `IX_Users_DateOfBirth` . . . 240
10.3 UserContact - Jakub Kaliński . . . 240

10.3.1 Index: IX_UserContact_Email . . . 240
10.4 UserAddressDetails - Jakub Kaliński . . . 241
10.4.1 Index: IX_UserAddressDetails_PostalCode . . . 241
10.4.2 Index: IX_UserAddressDetails_LocationID . . . 241
10.5 Studies - Michał Szymocha . . . 241
10.5.1 Index: IX_Studies_EnrollmentDeadline . . . 241
10.6 Subject - Michał Szymocha . . . 241
10.6.1 Index: IX_Subject_SubjectCoordinatorID . . . 241
10.7 SemesterDetails - Michał Szymocha . . . 241
10.7.1 Index: IX_SemesterDetails_StudiesID . . . 241
10.7.2 Index: IX_SemesterDetails_StartDate . . . 241
10.7.3 Index: IX_SemesterDetails_EndDate . . . 241
10.8 ClassMeeting - Michał Szymocha . . . 242
10.8.1 Index: IX_ClassMeeting_SubjectID . . . 242
10.8.2 Index: IX_ClassMeeting_TeacherID . . . 242
10.8.3 Index: IX_ClassMeeting_TranslatorID . . . 242
10.8.4 Index: IX_ClassMeeting_LanguageID . . . 242
10.9 StationaryClass - Michał Szymocha . . . 242
10.9.1 Index: IX_StationaryClass_StartDate . . . 242
10.9.2 Index: IX_StationaryClass_RoomID . . . 242
10.10 StationaryMeeting - Emil Żychowicz . . . 242
10.10.1 Index: IX_StationaryMeeting_Module . . . 242
10.10.2 Index: IX_StationaryMeeting_Teacher . . . 242
10.10.3 Index: IX_StationaryMeeting_Date . . . 243
10.11 StationaryMeetingDetails - Emil Żychowicz . . . 243
10.11.1 Index: IX_StationaryMeetingDetails_Participant . . . 243
10.12 OnlineLiveMeeting - Emil Żychowicz . . . 243
10.12.1 Index: IX_OnlineLiveMeeting_Module . . . 243
10.12.2 Index: IX_OnlineLiveMeeting_Teacher . . . 243
10.12.3 Index: IX_OnlineLiveMeeting_Date . . . 243
10.13 OnlineLiveMeetingDetails - Emil Żychowicz . . . 243
10.13.1 Index: IX_OnlineLiveMeetingDetails_Participant . . . 243
10.14 OfflineVideo - Emil Żychowicz . . . 243
10.14.1 Index: IX_OfflineVideo_Module . . . 243
10.14.2 Index: IX_OfflineVideo_Teacher . . . 244
10.15 OfflineVideoDetails - Emil Żychowicz . . . 244
10.15.1 Index: IX_OfflineVideoDetails_Participant . . . 244
10.16 Courses - Emil Żychowicz . . . 244
10.16.1 Index: IX_Courses_ServiceID . . . 244
10.16.2 Index: IX_Courses_CourseCoordinatorID . . . 244
10.16.3 Index: IX_Courses_CourseDate . . . 244
10.16.4 Index: IX_Courses_CourseName . . . 244
10.17 Modules - Emil Żychowicz . . . 244
10.17.1 Index: IX_Modules_CourseID . . . 244
10.17.2 Index: IX_Modules_ModuleCoordinatorID . . . 244
10.17.3 Index: IX_Modules_LanguageID . . . 245
10.17.4 Index: IX_Modules_TranslatorID . . . 245
10.18 CourseParticipants - Emil Żychowicz . . . 245
10.18.1 Index: IX_CourseParticipants_CourseID . . . 245
10.19 Payments - Emil Żychowicz . . . 245
10.19.1 Index: IX_Payments_OrderID . . . 245
10.20 Orders - Emil Żychowicz . . . 245
10.20.1 Index: IX_Orders_UserID . . . 245
10.20.2 Index: IX_Orders_OrderDate . . . 245
10.21 Employees - Jakub Kaliński . . . 245
10.21.1 Index: IX_Employees_DateOfHire . . . 245
10.22 AsyncClassDetails - Michał Szymocha . . . 246
10.22.1 Index: IX_AsyncClassDetails_StudentID . . . 246
10.23 SyncClassDetails - Michał Szymocha . . . 246
10.23.1 Index: IX_SyncClassDetails_StudentID . . . 246

10.24 StudiesDetails - Michał Szymocha . . . 246
10.24.1 Index: IX_StudiesDetails_Studies_Semester . . . 246
10.25 Internship - Michał Szymocha . . . 246
10.25.1 Index: IX_Internship_StudiesID . . . 246
10.25.2 Index: IX_Internship_StartDate . . . 246
10.26 InternshipDetails - Michał Szymocha . . . 246
10.26.1 Index: IX_InternshipDetails_StudentID . . . 246
10.27 Convention - Michał Szymocha . . . 246
10.27.1 Index: IX_Convention_SemesterID . . . 246
10.27.2 Index: IX_Convention_StartDate . . . 247
10.28 OfflineVideoClass - Michał Szymocha . . . 247
10.28.1 Index: IX_OfflineVideoClass_StartDate . . . 247
10.28.2 Index: IX_OfflineVideoClass_Deadline . . . 247
10.29 OnlineLiveClass - Michał Szymocha . . . 247
10.29.1 Index: IX_OnlineLiveClass_StartDate . . . 247
10.30 EmployeesSuperior - Jakub Kaliński . . . 247
10.30.1 Index: IX_EmployeesSuperior_ReportsTo . . . 247
10.31 UserTypePermissionsHierarchy - Jakub Kaliński . . . 247
10.31.1 Index: IX_UserTypePermissionsHierarchy_DirectSupervisor . . . 247
10.32 Webinars - Jakub Kaliński - Jakub Kaliński . . . 247
10.32.1 Index: IX_Webinars_TeacherID . . . 247
10.32.2 Index: IX_Webinars_TranslatorID . . . 248
10.32.3 Index: IX_Webinars_LanguageID . . . 248
10.33 WebinarDetails - Jakub Kaliński . . . 248
10.33.1 Index: IX_WebinarDetails_UserID . . . 248
10.34 Poprawa wydajności uzyskana za pomocą indeksów . . . 248
10.35 Statystyki fragmentacji dla indeksów . . . 248

1 Funkcje i użytkownicy systemu

1.1 Model uprawnień w systemie

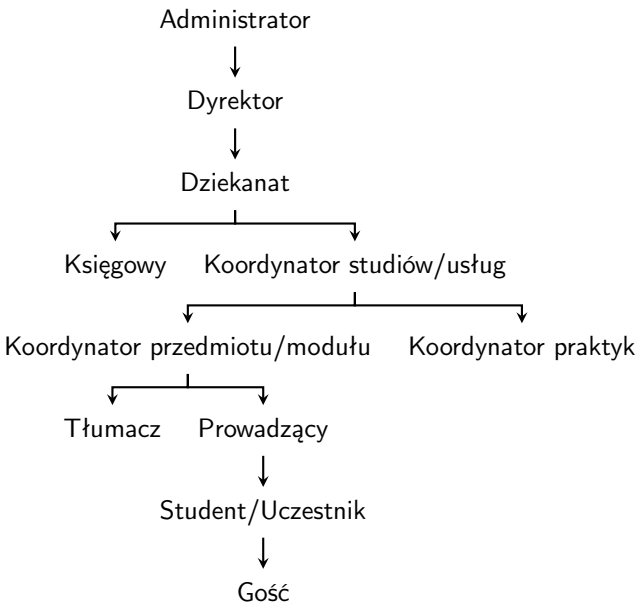
Uprawnienia w systemie są hierarchiczne, co oznacza, że użytkownicy z wyższym poziomem uprawnień automatycznie posiadają dostęp do funkcji przysługujących użytkownikom na niższych poziomach. Przypisana do każdej funkcji rola określa najniższy poziom uprawnień, jaki musi posiadać użytkownik, aby wykonać określone działanie. Jeśli przy funkcji wskazano dwie role, oznacza to, że minimalny poziom uprawnień jest równy niższej z tych dwóch ról. Jednocześnie wskazanie dwóch ról precyzuje, kto najczęściej będzie korzystać z tej funkcji. Taki model uprawnień umożliwia zarządzanie systemem w sposób kompleksowy i elastyczny.

Przypisana do danej funkcji rola wskazuje głównego użytkownika, który będzie z tej funkcji korzystać w standardowych warunkach. Choć wszyscy użytkownicy posiadający wyższe uprawnienia również mogą realizować te działania, jest to przewidziane jedynie dla sytuacji nadzwyczajnych lub wyjątkowych, wymagających ich interwencji. W ten sposób system pozostaje przejrzysty i unika niepotrzebnego angażowania osób o wyższych uprawnieniach w codzienne operacje.

1.2 Użytkownicy systemu (z odwzorowaniem hierarchii)

1. Administrator
2. Dyrektor
3. Dziekanat
   (a) Koordynator studiów/usług
      i. Koordynator praktyk
      ii. Koordynator przedmiotu/modułu
         A. Prowadzący
            - Student/Uczestnik
            - Gość
         B. Tłumacz
   (b) Księgowy

1.2.1 Schemat użytkowników systemu



1.3 Funkcje systemu

1.3.1 Konta i uprawnienia

- **Dodawanie/usuwanie administratora** – [Dyrektor]
- **Dodawanie/usuwanie dziekanatu** – [Dyrektor]
- **Dodawanie/usuwanie koordynatora studiów** – [Dziekanat]
- **Dodawanie/usuwanie koordynatora przedmiotu** – [Dziekanat]
- **Dodawanie/usuwanie prowadzącego** – [Koordynator przedmiotu]
- **Dodawanie/usuwanie tłumacza** – [Koordynator przedmiotu]
- **Dodawanie/usuwanie księgowego** – [Dziekanat]
- **Dodawanie/usuwanie koordynatora praktyk** – [Koordynator studiów]
- **Zakładanie/usuwanie konta studenta** – [Student]
- **Zakładanie/usuwanie konta gościa** *(konto tymczasowe z ograniczonymi uprawnieniami)* – [Gość]

1.3.2 Webinary

- **Dodawanie/usuwanie webinaru** – [Koordynator przedmiotu]
- **Edycja webinaru** – [Prowadzący]
- **Przypisywanie prowadzących** – [Koordynator przedmiotu]
- **Przypisywanie tłumaczy** – [Prowadzący]
- **Dodawanie/usuwanie/modyfikacja tłumaczenia** – [Tłumacz]
- **Zarządzanie harmonogramem webinaru** – [Prowadzący]
- **Zapisywanie uczestników na webinar** *(po potwierdzeniu opłacenia – dotyczy jedynie płatnych webinarów)* – [Student]
- **Umożliwienie obecności bez opłaty** – [Koordynator studiów]
- **Przeglądanie listy webinarów** – [Gość]
- **Dostęp do szczegółów webinaru** – [Zapisany student], [Prowadzący]

1.3.3 Kursy

- **Przegląd podstawowych informacji o kursie** *(moduły, cena, okres, język prowadzenia)* – [Gość], [Student]
- **Przegląd szczegółowych informacji o kursie** *(harmonogram, lista prowadzących)* – [Student], [Koordynator studiów], [Prowadzący]
- **Dodawanie/usuwanie modułów** – [Koordynator przedmiotu]
- **Modyfikacja modułów** – [Koordynator przedmiotu]
- **Dodawanie/usuwanie kursu** – [Koordynator studiów]
- **Modyfikacja kursu** *(cena, termin, limit miejsc)* – [Koordynator studiów]
- **Zapisywanie frekwencji podczas modułu kursu** – [Prowadzący]
- **Wyznaczanie statusu zaliczenia kursu** – [System]
- **Sprawdzanie dostępności kursu** *(czy limit miejsc nie został osiągnięty)* – [System], [Gość]
- **Zapis uczestnika na kurs** – [Student]

- **Zatwierdzenie zapisu uczestnika na kurs** *(po potwierdzeniu opłacenia kursu i przy dostępności miejsc)* – [System]
- **Udzielenie dostępu do kursu** – [System]
- **Generowanie dyplomu ukończenia kursu** – [Koordynator studiów]
- **Udostępnienie kursu tłumaczowi** – [Koordynator przedmiotu]
- **Dodawanie/usuwanie/modyfikacja tłumaczenia** – [Tłumacz]
- **Sprawdzanie stanu płatności za kurs** – [System], [Księgowy]
- **Monitorowanie postępu uczestnika** – [Koordynator przedmiotu], [Koordynator studiów]
- **Zmiana zasad cen i płatności dla konkretnych uczestników** – [Koordynator studiów]

1.3.4 Studia

- **Dodawanie/usuwanie studiów** – [Koordynator studiów]
- **Edycja sylabusa studiów** *(przed rozpoczęciem)* – [Koordynator studiów]
- **Dodawanie/usuwanie przedmiotów** – [Koordynator studiów]
- **Modyfikacja przedmiotów** – [Koordynator studiów]
- **Dodawanie/usuwanie praktyk** – [Koordynator praktyk]
- **Zarządzanie harmonogramem praktyk** – [Koordynator praktyk]
- **Sprawdzanie obecności na praktykach** – [Koordynator praktyk]
- **Modyfikacja praktyk** *(ich programu)* – [Koordynator praktyk]
- **Modyfikacja harmonogramu studiów** – [Koordynator studiów]
- **Przypisywanie tłumacza do przedmiotu** – [Koordynator przedmiotu]
- **Dodawanie/usuwanie/modyfikacja tłumaczenia** – [Tłumacz]
- **Zapisywanie na studia** – [Student], [Gość]
- **Zapisywanie na pojedyncze spotkania** – [Gość], [Student spoza studiów]
- **Monitorowanie frekwencji studentów** – [Prowadzący], [Koordynator przedmiotu]
- **Przypisywanie alternatywnych kursów do odrobienia zajęć** – [Prowadzący]
- **Weryfikacja statusu praktyk** – [Koordynator praktyk]
- **Generowanie zaświadczenia o ukończeniu praktyk** – [Koordynator praktyk]
- **Generowanie dyplomu ukończenia studiów** – [Koordynator studiów]
- **Wyliczanie wolnych miejsc** – [System]
- **Zmiana zasad cen i płatności dla konkretnych uczestników** – [Koordynator studiów]

1.3.5 Koszyk – system płatności

- **Dodawanie produktów do koszyka** *(Produkty – [Webinar], [Kurs], [Studia], pojedyncze wydarzenia)* – [Student], [Gość]
- **Usuwanie produktów z koszyka** – [Student], [Gość]
- **Wyliczanie wartości koszyka** – [System]
- **Tworzenie zamówienia** – [Student], [Gość]
- **Rejestracja zamówienia** – [System]
- **Generowanie linku do płatności** – [System] *za pomocą systemu płatności*
- **Rejestrowanie wpłaty** – [System] *za pomocą systemu płatności*
- **Rejestrowanie częściowej wpłaty** *(zaliczki)* – [System] *za pomocą systemu płatności*
- **Obsługa płatności ratalnych** *(raty)* – [System]
- **Zmiana zasad płatności i rabatów dla uczestników** – [Koordynator studiów]
- **Monitorowanie statusu płatności uczestników** – [Księgowy], [System]
- **Zmiana statusu płatności** – [System]
- **Monitorowanie zaległości w płatnościach** – [Księgowy], [System]

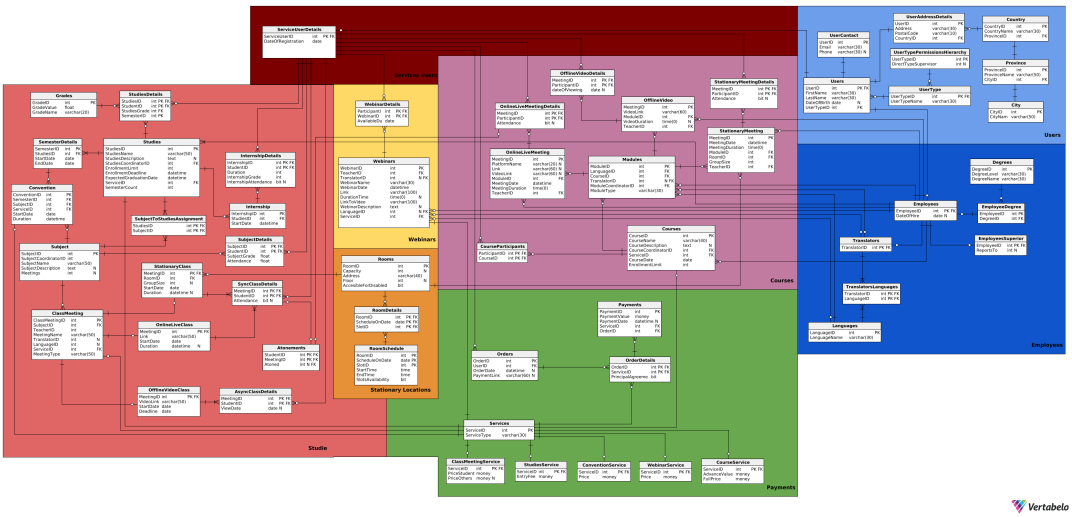
1.3.6 Raporty

- **Raport finansowy** *(przychody z [webinarów], [kursów], [studiów])* – [Księgowy], [Dziekanat]
- **Lista dłużników** *(osoby, które skorzystały z usług ale nie zapłaciły całej kwoty)* – [Księgowy], [Dziekanat]
- **Raport liczby zapisanych uczestników na przyszłe wydarzenia** *(Z podziałem na typ wydarzenia - [Webinar], [Kurs] i [Studia], oraz formę - [stacjonarne], [niestacjonarne], [hybrydowe])* – [Dziekanat]
- **Raport frekwencji na zakończonych wydarzeniach** – [Koordynator przedmiotu], [Koordynator studiów]
- **Lista obecności na wydarzeniach** – [Prowadzący], [Dziekanat]
- **Raport kolizji** *(osoby zapisane na kolidujące wydarzenia)* – [Dziekanat]
- **Lista stałych klientów** – [Dziekanat]
- **Wykaz zapisów uczestnika** – [Student]
- **Wykaz wydarzeń prowadzonych przez prowadzącego** – [Prowadzący]
- **Wykaz wydarzeń tłumaczonych przez tłumacza** – [Tłumacz]

1.3.7 Dodatkowe funkcje systemu

- **Automatyczne, cykliczne wykonywanie backupów** – [System]
- **Wykonywanie backupów na żądanie** – [Administrator]
- **Odtwarzanie bazy danych z backupów** – [Administrator]
- **Wysyłanie przypomnień o zaległych płatnościach** – [System]
- **Wysyłanie powiadomień o zmianach w harmonogramach** – [System]
- **Wysyłanie dyplomów pocztą** – [Dziekanat]

2 Schemat bazy danych



Rysunek 1: Schemat bazy.

3 Generowanie danych testowych do bazy SQL

Do generowania danych testowych napisaliśmy skrypt w języku Python. Jego podstawą jest biblioteka Faker, która pozwala na generowanie losowych, realistycznych danych. Jest on dostępny w repozytorium GitHub pod adresem https://github.com/ezychowicz/Introduction-to-Databases

3.1 Biblioteki i konfiguracje

- **Faker** — generowanie losowych, realistycznych danych, np. nazw, adresów, dat.
- **random** — wprowadzenie losowości w doborze danych.
- **datetime** i **timedelta** — manipulacja datami i czasem.

3.2 Generowane dane

- **Użytkownicy (Users):** Podział na typy: studenci, wykładowcy, tłumacze.
- **Studenci (Students):** Podtyp użytkowników z danymi dodatkowymi tylko dla studentów.
- **Pracownicy (Employees):** Podtyp użytkowników z danymi dodatkowymi tylko dla pracowników.
- **Lokalizacje i adresy:** Kraje, stany, miasta, dokładne adresy.
- **Studia i przedmioty:** Tworzenie studiów, kursów i webinarów z przypisaniem koordynatorów i studentów, a także ocen dla uczestników.
- **Spotkania i zajęcia:** Różne typy spotkań, takie jak stacjonarne, online czy wideo na żądanie.
- **System płatności:** Usługi, zamówienia i płatności, w tym różne ceny dla różnych grup użytkowników.

3.3 Struktura i logika

- Dane są generowane zgodnie z ustalonymi relacjami między encjami (np. użytkownicy → studenci → studia → przedmioty). Kolejne kolekcje są generowane zgodnie z porządkiem topologicznym.
- Ustalono limity, takie jak liczba użytkowników, języków, lokalizacji, przedmiotów, kursów. Pozwala to na szybkie generowanie danych testowych o różnym rozmiarze.

- Wprowadzono parametryzowane opcje generowania danych, takie jak m.in. liczba użytkowników, liczba przedmiotów czy zakres dat. Pozwala to zachować sensowność i spójność danych w bazie.
- Do zachowania spójności relacji wykorzystujemy Pythonowe słowniki, które przechowują kolekcje elementów wcześniejszych w porządku topologicznym.

3.4 Generowanie danych do bazy SQL

- Na początku skrypt usuwa dane ze wszystkich tabel, aby zapewnić czystość bazy.
- Następnie generuje dane dla każdej z tabel, zaczynając od tych, które nie mają zależności.
- Na koniec wypisuje zapytania SQL typu `INSERT INTO`, które mogą być bezpośrednio wprowadzone do bazy danych.

3.5 Specjalne funkcje

- `quote_str` — obsługuje apostrofy w ciągach znaków.
- `format_date` i `format_datetime` — formatowanie dat w stylu `YYYY-MM-DD` lub `YYYY-MM-DD HH:MM:SS`.

4 Opisy tabel

4.1 Sekcja studiów

4.1.1 Tabela AsyncClassDetails

Zawiera szczegółowe informacje o asynchronicznych zajęciach zdalnych:

- **MeetingID** int – identyfikator spotkania, część klucza głównego
- **StudentID** int – identyfikator studenta, część klucza głównego
- **ViewDate** date nullable – data obejrzenia materiału

```
CREATE TABLE AsyncClassDetails (
    MeetingID int NOT NULL,
    StudentID int NOT NULL,
    ViewDate date NULL,
    CONSTRAINT AsyncClassDetails_pk PRIMARY KEY (MeetingID, StudentID)
);
```

Relacje inicjalizowane przez tą tabelę:

- Relacja do tabeli `OfflineVideoClass`:

```
ALTER TABLE AsyncClassDetails ADD CONSTRAINT AsyncClassDetails_OfflineVideoClass
    FOREIGN KEY (MeetingID)
    REFERENCES OfflineVideoClass (MeetingID);
```

- Relacja do tabeli `ServiceUserDetails`:

```
ALTER TABLE AsyncClassDetails ADD CONSTRAINT AsyncClassDetails_Users
    FOREIGN KEY (StudentID)
    REFERENCES ServiceUserDetails (ServiceUserID);
```

4.1.2 Tabela Atonements

Zawiera informacje o odrabianiu nieobecności kursami komercyjnymi:

- **StudentID** int – identyfikator studenta, część klucza głównego
- **MeetingID** int – identyfikator odrabianego spotkania, część klucza głównego
- **Atoned** int - identyfikator spotkania w ramach kursu komercyjnego

```
CREATE TABLE Atonements (
    StudentID int NOT NULL,
    MeetingID int NOT NULL,
    Atoned int NOT NULL,
    CONSTRAINT Atonements_pk PRIMARY KEY (StudentID, MeetingID)
);
```

Relacje inicjalizowane przez tą tabelę:

- Relacja do tabeli `OnlineLiveMeetingDetails`:

```
ALTER TABLE Atonements ADD CONSTRAINT Atonements_OnlineLiveMeetingDetails
    FOREIGN KEY (Atoned,StudentID)
    REFERENCES OnlineLiveMeetingDetails (MeetingID,ParticipantID);
```

- Relacja do tabeli `SyncClassDetails`:

```
ALTER TABLE Atonements ADD CONSTRAINT Atonements_SyncClassDetails
    FOREIGN KEY (MeetingID,StudentID)
    REFERENCES SyncClassDetails (MeetingID,StudentID);
```

4.1.3 Tabela ClassMeeting

Zawiera informacje o spotkaniach w ramach studiów:

- **ClassMeetingID** int – klucz główny, identyfikator spotkania
- **SubjectID** int – identyfikator przedmiotu
- **TeacherID** int – identyfikator nauczyciela
- **MeetingName** varchar(50) – nazwa spotkania
- **TranslatorID** int nullable – identyfikator tłumacza
- **LanguageID** int nullable – identyfikator języka
- **ServiceID** int – identyfikator usługi
- **MeetingType** varchar(50) – typ spotkania
- **DurationTime** time(0) nullable – czas trwania spotkania, warunek: DurationTime > '00:00:00'

```
CREATE TABLE ClassMeeting (
    ClassMeetingID int NOT NULL,
    SubjectID int NOT NULL,
    TeacherID int NOT NULL,
    MeetingName varchar(50) NOT NULL,
    TranslatorID int NULL,
    LanguageID int NULL,
    ServiceID int NOT NULL,
    MeetingType varchar(50) NOT NULL,
    DurationTime time(0) NULL CHECK (DurationTime > '00:00:00'),
    CONSTRAINT ClassMeeting_pk PRIMARY KEY (ClassMeetingID)
);
```

Relacje inicjalizowane przez tą tabelę:

- Relacja do tabeli `Services`:

```
ALTER TABLE ClassMeeting ADD CONSTRAINT ClassMeeting_Services
    FOREIGN KEY (ServiceID)
    REFERENCES Services (ServiceID);
```

- Relacja do tabeli `Subject`:

```
ALTER TABLE ClassMeeting ADD CONSTRAINT ClassMeeting_Subject
    FOREIGN KEY (SubjectID)
    REFERENCES Subject (SubjectID);
```

Relacje odnoszące się do tej tabeli:

- Relacja inicjowana przez tabelę `OfflineVideoClass`:

```
ALTER TABLE OfflineVideoClass ADD CONSTRAINT OnlineAsyncMeeting_ClassMeeting
    FOREIGN KEY (MeetingID)
    REFERENCES ClassMeeting (ClassMeetingID);
```

- Relacja inicjowana przez tabelę `OnlineLiveClass`:

```
ALTER TABLE OnlineLiveClass ADD CONSTRAINT OnlineSyncMeeting_ClassMeeting
    FOREIGN KEY (MeetingID)
    REFERENCES ClassMeeting (ClassMeetingID);
```

- Relacja inicjowana przez tabelę `StationaryClass`:

```
ALTER TABLE StationaryClass ADD CONSTRAINT StationaryMeeting_ClassMeeting
    FOREIGN KEY (MeetingID)
    REFERENCES ClassMeeting (ClassMeetingID);
```

4.1.4 Tabela Convention

Zawiera informacje o zjazdach w danym semestrze:

- **SemesterID** int – identyfikator semestru
- **SubjectID** int – część klucza głównego, identyfikator przedmiotu
- **ConventionID** int - część klucza głównego, identyfikator zjazdu
- **ServiceID** int – identyfikator usługi
- **StartDate** date – data rozpoczęcia
- **Duration** int – długość zjazdu w dniach, wartość domyślna: 3, warunek: Duration > 0

```
CREATE TABLE Convention (
    SemesterID int NOT NULL,
    SubjectID int NOT NULL,
    ConventionID int NOT NULL,
    ServiceID int NOT NULL,
    StartDate date NOT NULL,
    Duration int NOT NULL DEFAULT 3 CHECK (Duration > 0),
    CONSTRAINT Convention_pk PRIMARY KEY (SubjectID, ConventionID)
);
```

Relacje inicjalizowane przez tą tabelę:

- Relacja do tabeli `SemesterDetails`:

```
ALTER TABLE Convention ADD CONSTRAINT Convention_SemesterDetails
    FOREIGN KEY (SemesterID)
    REFERENCES SemesterDetails (SemesterID);
```

- Relacja do tabeli `Services`:

```
ALTER TABLE Convention ADD CONSTRAINT Convention_Services
    FOREIGN KEY (ServiceID)
    REFERENCES Services (ServiceID);
```

- Relacja do tabeli `Subject`:

```
ALTER TABLE Convention ADD CONSTRAINT Subject_Convention
    FOREIGN KEY (SubjectID)
    REFERENCES Subject (SubjectID);
```

4.1.5 Tabela Grades

Słownik ocen:

- **GradeID** int – klucz główny, identyfikator oceny
- **GradeValue** real – wartość oceny, warunek: GradeValue > 0
- **GradeName** varchar(20) – nazwa oceny

```
CREATE TABLE Grades (
    GradeID int NOT NULL,
    GradeValue real NOT NULL CHECK (GradeValue > 0),
    GradeName varchar(20) NOT NULL,
    CONSTRAINT Grades_pk PRIMARY KEY (GradeID)
);
```

Relacje odnoszące się do tej tabeli:

- Relacja inicjowana przez tabelę `StudiesDetails`:

```
ALTER TABLE StudiesDetails ADD CONSTRAINT StudiesDetails_Grades
    FOREIGN KEY (StudiesGrade)
    REFERENCES Grades (GradeID);
```

4.1.6 Tabela Internship

Zawiera informacje o praktykach:

- **InternshipID** int – klucz główny, identyfikator praktyk
- **StudiesID** int – identyfikator studiów
- **StartDate** datetime – data rozpoczęcia praktyk

```
CREATE TABLE Internship (
    InternshipID int NOT NULL,
    StudiesID int NOT NULL,
    StartDate datetime NOT NULL CHECK StartDate > '2015-01-01' AND StartDate <
        '2030-01-01',
    CONSTRAINT Internship_pk PRIMARY KEY (InternshipID)
);
```

Relacje inicjalizowane przez tą tabelę:

- Relacja do tabeli `Studies`:

```
ALTER TABLE Internship ADD CONSTRAINT Internship_Studies
    FOREIGN KEY (StudiesID)
    REFERENCES Studies (StudiesID);
```

Relacje odnoszące się do tej tabeli:

- Relacja inicjowana przez tabelę `InternshipDetails`:

```
ALTER TABLE InternshipDetails ADD CONSTRAINT InternshipDetails_Internship
    FOREIGN KEY (InternshipID)
    REFERENCES Internship (InternshipID);
```

4.1.7 Tabela InternshipDetails

Zawiera szczegółowe informacje dotyczące praktyk dla każdego studenta:

- **InternshipID** int – część klucza głównego, identyfikator praktyk
- **StudentID** int – część klucza głównego, identyfikator studenta
- **Duration** int – długość praktyk w dniach, wartość domyślna: 30, warunek: Duration > 0
- **InternshipGrade** int – ocena praktyk
- **InternshipAttendance** bit nullable – frekwencja na praktykach, wartość domyślna: 0

```
CREATE TABLE InternshipDetails (
    InternshipID int NOT NULL,
    StudentID int NOT NULL,
    Duration int NOT NULL DEFAULT 30 CHECK (Duration > 0),
    InternshipGrade int NOT NULL,
    InternshipAttendance bit NULL DEFAULT 0,
    CONSTRAINT InternshipDetails_pk PRIMARY KEY (InternshipID, StudentID)
);
```

Relacje inicjalizowane przez tą tabelę:

- Relacja do tabeli `Internship`:

```
ALTER TABLE InternshipDetails ADD CONSTRAINT InternshipDetails_Internship
    FOREIGN KEY (InternshipID)
    REFERENCES Internship (InternshipID);
```

- Relacja do tabeli `ServiceUserDetails`:

```
ALTER TABLE InternshipDetails ADD CONSTRAINT InternshipDetails_Users
    FOREIGN KEY (StudentID)
    REFERENCES ServiceUserDetails (ServiceUserID);
```

4.1.8 Tabela OfflineVideoClass

Zawiera informacje o zajęciach polegających na obejrzeniu wideo w domu:

- **MeetingID** int – klucz główny, identyfikator spotkania
- **VideoLink** varchar(50) – link do wideo
- **StartDate** date – data rozpoczęcia
- **Deadline** date – termin zakończenia, warunek: Deadline > StartDate

```
CREATE TABLE OfflineVideoClass (
    MeetingID int NOT NULL,
    VideoLink varchar(50) NOT NULL,
    StartDate date NOT NULL,
    Deadline date NOT NULL CHECK (Deadline > StartDate),
    CONSTRAINT OfflineVideoClass_pk PRIMARY KEY (MeetingID)
);
```

Relacje inicjalizowane przez tą tabelę:

- Relacja do tabeli `ClassMeeting`:

```
ALTER TABLE OfflineVideoClass ADD CONSTRAINT OnlineAsyncMeeting_ClassMeeting
    FOREIGN KEY (MeetingID)
    REFERENCES ClassMeeting (ClassMeetingID);
```

Relacje odnoszące się do tej tabeli:

- Relacja inicjowana przez tabelę `AsyncClassDetails`:

```
ALTER TABLE AsyncClassDetails ADD CONSTRAINT AsyncClassDetails_OfflineVideoClass
    FOREIGN KEY (MeetingID)
    REFERENCES OfflineVideoClass (MeetingID);
```

4.1.9 Tabela OnlineLiveClass

Zawiera informacje o zajęciach online na żywo:

- **MeetingID** int – klucz główny, identyfikator spotkania
- **Link** varchar(50) – link do spotkania
- **StartDate** datetime – data rozpoczęcia
- **Duration** time(0) nullable – czas trwania, wartość domyślna: 01:30:00

```
CREATE TABLE OnlineLiveClass (
    MeetingID int NOT NULL,
    Link varchar(50) NOT NULL,
    StartDate datetime NOT NULL,
    Duration time(0) NULL DEFAULT '01:30:00',
    CONSTRAINT OnlineLiveClass_pk PRIMARY KEY (MeetingID)
);
```

Relacje inicjalizowane przez tą tabelę:

- Relacja do tabeli `ClassMeeting`:

```
ALTER TABLE OnlineLiveClass ADD CONSTRAINT OnlineSyncMeeting_ClassMeeting
    FOREIGN KEY (MeetingID)
    REFERENCES ClassMeeting (ClassMeetingID);
```

Relacje odnoszące się do tej tabeli:

- Relacja inicjowana przez tabelę `SyncClassDetails`:

```
ALTER TABLE SyncClassDetails ADD CONSTRAINT SyncClassDetails_OnlineLiveClass
    FOREIGN KEY (MeetingID)
    REFERENCES OnlineLiveClass (MeetingID);
```

4.1.10 Tabela SemesterDetails

Łącznik między zjazdami, studiami i przedmiotami:

- **SemesterID** int – klucz główny, identyfikator semestru
- **StudiesID** int – identyfikator studiów

```
CREATE TABLE SemesterDetails (
    SemesterID int NOT NULL,
    StudiesID int NOT NULL,
    StartDate date NOT NULL,
    EndDate date NOT NULL CHECK (EndDate > StartDate),
    CONSTRAINT Semester_pk PRIMARY KEY (SemesterID)
);
```

Relacje inicjalizowane przez tą tabelę:

- Relacja do tabeli `Studies`:

```
ALTER TABLE SemesterDetails ADD CONSTRAINT SemesterDetails_Studies
    FOREIGN KEY (StudiesID)
    REFERENCES Studies (StudiesID);
```

Relacje odnoszące się do tej tabeli:

- Relacja inicjowana przez tabelę `Convention`:

```
ALTER TABLE Convention ADD CONSTRAINT Convention_SemesterDetails
    FOREIGN KEY (SemesterID)
    REFERENCES SemesterDetails (SemesterID);
```

- Relacja inicjowana przez tabelę `StudiesDetails`:

```
ALTER TABLE StudiesDetails ADD CONSTRAINT SemesterDetails_StudiesDetails
    FOREIGN KEY (SemesterID)
    REFERENCES SemesterDetails (SemesterID);
```

4.1.11 Tabela StationaryClass

Zawiera informacje o zajęciach stacjonarnych:

- **MeetingID** int – klucz główny, identyfikator spotkania
- **RoomID** int – identyfikator sali
- **GroupSize** int nullable – wielkość grupy, wartość domyślna: 30, warunek: GroupSize > 0
- **StartDate** date – data rozpoczęcia
- **Duration** datetime nullable – czas trwania
- **Limit** int – warunek: Limit > 0

```
CREATE TABLE StationaryClass (
    MeetingID int NOT NULL,
    RoomID int NOT NULL,
    GroupSize int NULL DEFAULT 30 CHECK (GroupSize > 0),
    StartDate datetime NOT NULL,
    Duration time(0) NULL,
    Limit int CHECK (Limit > 0),
    CONSTRAINT StationaryClass_pk PRIMARY KEY (MeetingID)
);
```

Relacje inicjalizowane przez tą tabelę:

- Relacja do tabeli `ClassMeeting`:

```
ALTER TABLE StationaryClass ADD CONSTRAINT StationaryMeeting_ClassMeeting
    FOREIGN KEY (MeetingID)
    REFERENCES ClassMeeting (ClassMeetingID);
```

- Relacja do tabeli `Rooms`:

```
ALTER TABLE StationaryClass ADD CONSTRAINT StudiesStationaryMeeting_Rooms
    FOREIGN KEY (RoomID)
    REFERENCES Rooms (RoomID);
```

Relacje odnoszące się do tej tabeli:

- Relacja inicjowana przez tabelę `SyncClassDetails`:

```
ALTER TABLE SyncClassDetails ADD CONSTRAINT StationaryClassDetails_StationaryClass
    FOREIGN KEY (MeetingID)
    REFERENCES StationaryClass (MeetingID);
```

4.1.12 Tabela Studies

Zawiera informacje o poszczególnych studiach:

- **StudiesID** int – klucz główny, identyfikator studiów
- **StudiesName** varchar(50) – nazwa studiów
- **StudiesDescription** varchar(255) nullable – opis studiów
- **StudiesCoordinatorID** int – identyfikator koordynatora studiów
- **EnrollmentLimit** int – limit zapisów na studia, wartość domyślna: 100
- **EnrollmentDeadline** datetime – ostateczny termin zapisów
- **ExpectedGraduationDate** datetime – przewidywana data ukończenia studiów
- **ServiceID** int – identyfikator usługi
- **SemesterCount** int – liczba semestrów

```
CREATE TABLE Studies (
    StudiesID int NOT NULL,
    StudiesName varchar(50) NOT NULL,
    StudiesDescription varchar(255) NULL,
    StudiesCoordinatorID int NOT NULL,
    EnrollmentLimit int NOT NULL DEFAULT 100 CHECK (EnrollmentLimit > 0),
    EnrollmentDeadline datetime NOT NULL,
    ExpectedGraduationDate datetime NOT NULL CHECK (ExpectedGraduationDate >
        EnrollmentDeadline),
    ServiceID int NOT NULL,
    SemesterCount int NOT NULL DEFAULT 7,
    CONSTRAINT Studies_pk PRIMARY KEY (StudiesID)
);
```

Relacje inicjalizowane przez tą tabelę:

- Relacja do tabeli `Employees`:

```
ALTER TABLE Studies ADD CONSTRAINT Employees_Studies
    FOREIGN KEY (StudiesCoordinatorID)
    REFERENCES Employees (EmployeeID);
```

- Relacja do tabeli `Services`:

```
ALTER TABLE Studies ADD CONSTRAINT Studies_Services
    FOREIGN KEY (ServiceID)
    REFERENCES Services (ServiceID);
```

Relacje odnoszące się do tej tabeli:

- Relacja inicjowana przez tabelę `Internship`:

```
ALTER TABLE Internship ADD CONSTRAINT Internship_Studies
    FOREIGN KEY (StudiesID)
    REFERENCES Studies (StudiesID);
```

- Relacja inicjowana przez tabelę `SemesterDetails`:

```
ALTER TABLE SemesterDetails ADD CONSTRAINT SemesterDetails_Studies
    FOREIGN KEY (StudiesID)
    REFERENCES Studies (StudiesID);
```

- Relacja inicjowana przez tabelę `StudiesDetails`:

```
ALTER TABLE StudiesDetails ADD CONSTRAINT StudiesDetails_Studies
    FOREIGN KEY (StudiesID)
    REFERENCES Studies (StudiesID);
```

- Relacja inicjowana przez tabelę `SubjectToStudiesAssignment`:

```
ALTER TABLE SubjectToStudiesAssignment ADD CONSTRAINT SubjectDetails2_Studies
    FOREIGN KEY (StudiesID)
    REFERENCES Studies (StudiesID);
```

4.1.13 Tabela StudiesDetails

Zawiera szczegółowe informacje o uczestnictwie w studiach:

- **StudiesID** int – część klucza głównego, identyfikator studiów
- **StudentID** int – część klucza głównego, identyfikator studenta
- **StudiesGrade** int – ocena studenta za studia
- **SemesterID** int - część klucza głównego, identyfikator semestru

```
CREATE TABLE StudiesDetails (
    StudiesID int NOT NULL,
    StudentID int NOT NULL,
    StudiesGrade int NOT NULL,
    SemesterID int NOT NULL,
    CONSTRAINT StudiesDetails_pk PRIMARY KEY (StudentID, StudiesID, SemesterID)
);
```

Relacje inicjalizowane przez tą tabelę:

- Relacja do tabeli `SemesterDetails`:

```
ALTER TABLE StudiesDetails ADD CONSTRAINT SemesterDetails_StudiesDetails
    FOREIGN KEY (SemesterID)
    REFERENCES SemesterDetails (SemesterID);
```

- Relacja do tabeli `Grades`:

```
ALTER TABLE StudiesDetails ADD CONSTRAINT StudiesDetails_Grades
    FOREIGN KEY (StudiesGrade)
    REFERENCES Grades (GradeID);
```

- Relacja do tabeli `Studies`:

```
ALTER TABLE StudiesDetails ADD CONSTRAINT StudiesDetails_Studies
    FOREIGN KEY (StudiesID)
    REFERENCES Studies (StudiesID);
```

- Relacja do tabeli `ServiceUserDetails`:

```
ALTER TABLE StudiesDetails ADD CONSTRAINT StudiesDetails_Users
    FOREIGN KEY (StudentID)
    REFERENCES ServiceUserDetails (ServiceUserID);
```

4.1.14 Tabela Subject

Zawiera informacje o przedmiotach:

- **SubjectID** int – klucz główny, identyfikator przedmiotu
- **StudiesID** int – identyfikator studiów
- **SubjectCoordinatorID** int – identyfikator koordynatora przedmiotu
- **SubjectName** varchar(50) – nazwa przedmiotu
- **SubjectDescription** varchar(255) nullable – opis przedmiotu
- **ServiceID** int – identyfikator usługi
- **Meetings** int - liczba spotkań w semestrze, warunek: Meetings > 0

```
CREATE TABLE Subject (
    SubjectID int NOT NULL,
    StudiesID int NOT NULL,
    SubjectCoordinatorID int NOT NULL,
    SubjectName varchar(50) NOT NULL,
    SubjectDescription varchar(255) NULL,
    ServiceID int NOT NULL,
    Meetings int NOT NULL CHECK (Meetings > 0),
    CONSTRAINT Subject_pk PRIMARY KEY (SubjectID)
);
```

Relacje odnoszące się do tej tabeli:

- Relacja inicjowana przez tabelę `ClassMeeting`:

```
ALTER TABLE ClassMeeting ADD CONSTRAINT ClassMeeting_Subject
    FOREIGN KEY (SubjectID)
    REFERENCES Subject (SubjectID);
```

- Relacja inicjowana przez tabelę `SubjectToStudiesAssignment`:

```
ALTER TABLE SubjectToStudiesAssignment ADD CONSTRAINT SubjectDetails2_Subject
    FOREIGN KEY (SubjectID)
    REFERENCES Subject (SubjectID);
```

- Relacja inicjowana przez tabelę `SubjectDetails`:

```
ALTER TABLE SubjectDetails ADD CONSTRAINT SubjectDetails_Subject
    FOREIGN KEY (SubjectID)
    REFERENCES Subject (SubjectID);
```

- Relacja inicjowana przez tabelę `Convention`:

```
ALTER TABLE Convention ADD CONSTRAINT Subject_Convention
    FOREIGN KEY (SubjectID)
    REFERENCES Subject (SubjectID);
```

4.1.15 Tabela SubjectDetails

Zawiera informacje o przedmiocie dla poszczególnych studentów:

- **SubjectID** int – część klucza głównego, identyfikator przedmiotu
- **StudentID** int – część klucza głównego, identyfikator studenta
- **SubjectGrade** int - ocena studenta z przedmiotu
- **Attendance** float - procentowa obecność studenta na zajęciach z przedmiotu
-

```
CREATE TABLE SubjectDetails (
    SubjectID int NOT NULL,
    StudentID int NOT NULL,
    SubjectGrade float NOT NULL,
    Attendance float NOT NULL CHECK (Attendance >= 0 AND Attendance <= 100),
    CONSTRAINT SubjectDetails_pk PRIMARY KEY (SubjectID, StudentID)
);
```

Relacje inicjalizowane przez tą tabelę:

- Relacja do tabeli `Subject`:

```
ALTER TABLE SubjectDetails ADD CONSTRAINT SubjectDetails_Subject
    FOREIGN KEY (SubjectID)
    REFERENCES Subject (SubjectID);
```

- Relacja do tabeli `ServiceUserDetails`:

```
ALTER TABLE SubjectDetails ADD CONSTRAINT SubjectDetails_Users
    FOREIGN KEY (StudentID)
    REFERENCES ServiceUserDetails (ServiceUserID);
```

4.1.16 Tabela SubjectStudiesBridge

Łącznik między przedmiotami a studiami:

- **StudiesID** int – część klucza głównego, identyfikator studiów
- **SubjectID** int – część klucza głównego, identyfikator przedmiotu

```
CREATE TABLE SubjectStudiesAssignment (
    StudiesID int NOT NULL,
    SubjectID int NOT NULL,
    CONSTRAINT SubjectStudiesAssignment_pk PRIMARY KEY (StudiesID, SubjectID)
);
```

4.1.17 Tabela SyncClassDetails

Zawiera szczegółowe informacje o uczestnictwie danego studenta w synchronicznych zajęciach online lub zajęciach stacjonarnych:

- **MeetingID** int – część klucza głównego, identyfikator spotkania
- **StudentID** int – część klucza głównego, identyfikator studenta
- **Attendance** bit nullable – obecność na zajęciach, wartość domyślna: 0

```
CREATE TABLE SyncClassDetails (
    MeetingID int NOT NULL,
    StudentID int NOT NULL,
    Attendance bit NULL DEFAULT 0,
    CONSTRAINT SyncClassDetails_pk PRIMARY KEY (MeetingID, StudentID)
);
```

Relacje inicjalizowane przez tą tabelę:

- Relacja do tabeli `StationaryClass`:

```
ALTER TABLE SyncClassDetails ADD CONSTRAINT StationaryClassDetails_StationaryClass
    FOREIGN KEY (MeetingID)
    REFERENCES StationaryClass (MeetingID);
```

- Relacja do tabeli `ServiceUserDetails`:

```
ALTER TABLE SyncClassDetails ADD CONSTRAINT StudyMeetingDetails_Users
    FOREIGN KEY (StudentID)
    REFERENCES ServiceUserDetails (ServiceUserID);
```

- Relacja do tabeli `OnlineLiveClass`:

```
ALTER TABLE SyncClassDetails ADD CONSTRAINT SyncClassDetails_OnlineLiveClass
    FOREIGN KEY (MeetingID)
    REFERENCES OnlineLiveClass (MeetingID);
```

Relacje odnoszące się do tej tabeli:

- Relacja inicjowana przez tabelę `Atonements`:

```
ALTER TABLE Atonements ADD CONSTRAINT Atonements_SyncClassDetails
    FOREIGN KEY (MeetingID,StudentID)
    REFERENCES SyncClassDetails (MeetingID,StudentID);
```

4.2 Sekcja webinarów

4.2.1 Tabela WebinarDetails

Zawiera szczegółowe informacje o uczestnictwie użytkowników w webinarach:

- **UserID** int – część klucza głównego, identyfikator użytkownika
- **WebinarID** int – część klucza głównego, identyfikator webinaru
- **AvailableDue** date – data wygaśnięcia dostępności

```
CREATE TABLE WebinarDetails (
    UserID int NOT NULL,
    WebinarID int NOT NULL,
    AvailableDue date NOT NULL CHECK (AvailableDue < '2030-01-01' AND AvailableDue >
        '2015-01-01')
    CONSTRAINT WebinarDetails_pk PRIMARY KEY (UserID, WebinarID)
);
```

Relacje inicjalizowane przez tą tabelę:

- Relacja do tabeli `ServiceUserDetails`:

```
ALTER TABLE WebinarDetails ADD CONSTRAINT WebinarDetails_Users
    FOREIGN KEY (ParticipantID)
    REFERENCES ServiceUserDetails (ServiceUserID);
```

- Relacja do tabeli `Webinars`:

```
ALTER TABLE WebinarDetails ADD CONSTRAINT WebinarDetails_Webinars
    FOREIGN KEY (WebinarID)
    REFERENCES Webinars (WebinarID);
```

4.2.2 Tabela Webinars

Zawiera szczegółowe informacje o webinarach:

- **WebinarID** int – klucz główny, identyfikator webinaru
- **TeacherID** int – identyfikator prowadzącego
- **TranslatorID** int nullable – identyfikator tłumacza
- **WebinarName** varchar(30) – nazwa webinaru
- **WebinarDate** datetime – data webinaru
- **Link** varchar(100) – link do webinaru
- **DurationTime** time(0) nullable – czas trwania webinaru
- **LinkToVideo** varchar(100) – link do nagrania
- **WebinarDescription** varchar(255) nullable – opis webinaru
- **LanguageID** int nullable – identyfikator języka
- **ServiceID** int – identyfikator usługi

```
CREATE TABLE Webinars (
    WebinarID int NOT NULL,
    TeacherID int NOT NULL,
    TranslatorID int NULL,
    WebinarName varchar(30) NOT NULL,
    WebinarDate datetime NOT NULL,
    Link varchar(100) NOT NULL,
    DurationTime time(0) NULL,
    LinkToVideo varchar(100) NOT NULL,
    WebinarDescription varchar(255) NULL,
    LanguageID int NULL,
    ServiceID int NOT NULL,
    CONSTRAINT Webinars_pk PRIMARY KEY (WebinarID)
);
```

Relacje inicjalizowane przez tą tabelę:

- Relacja do tabeli `Employees`:

```
ALTER TABLE Webinars ADD CONSTRAINT Webinars_Employees
    FOREIGN KEY (TeacherID)
    REFERENCES Employees (EmployeeID);
```

- Relacja do tabeli `Languages`:

```
ALTER TABLE Webinars ADD CONSTRAINT Webinars_Languages
    FOREIGN KEY (LanguageID)
    REFERENCES Languages (LanguageID);
```

- Relacja do tabeli `Services`:

```
ALTER TABLE Webinars ADD CONSTRAINT Webinars_Services
    FOREIGN KEY (ServiceID)
    REFERENCES Services (ServiceID);
```

- Relacja do tabeli `Translators`:

```
ALTER TABLE Webinars ADD CONSTRAINT Webinars_Translators
    FOREIGN KEY (TranslatorID)
    REFERENCES Translators (TranslatorID);
```

Relacje odnoszące się do tej tabeli:

- Relacja inicjowana przez tabelę `WebinarDetails`:

```
ALTER TABLE WebinarDetails ADD CONSTRAINT WebinarDetails_Webinars
    FOREIGN KEY (WebinarID)
    REFERENCES Webinars (WebinarID);
```

4.3 Sekcja kursów

4.3.1 Tabela Courses

Zawiera podstawowe informacje na temat kursów:

- **CourseID** int - klucz główny, identyfikator kursu;
- **CourseName** varchar(40) - nazwa kursu;
- **CourseDescription** text nullable - opis kursu;
- **CourseCoordinatorID** int - identyfikator koordynatora kursu;
- **ServiceID** int - identyfikator usługi, którą kurs stanowi;
- **CourseDate** date - data rozpoczęcia kursu;
- **EnrollmentLimit** int - maksymalna liczba osób, która może zapisać się na kurs. Musi być dodatni.

```
CREATE TABLE Courses (
    CourseID int  NOT NULL,
    CourseName varchar(40)  NOT NULL,
    CourseDescription varchar(255)  NULL,
    CourseCoordinatorID int  NOT NULL,
    ServiceID int  NOT NULL,
    CourseDate date  NOT NULL CHECK(CourseDate > '01-01-2015' AND CourseDate <
        '01-01-2030'),
    EnrollmentLimit int NOT NULL CHECK(EnrollmentLimit > 1),
    CONSTRAINT CourseID PRIMARY KEY  (CourseID)
);
```

Relacje inicjalizowane przez tą tabelę:

- Relacja do tabeli `Employees`:

```
ALTER TABLE Courses ADD CONSTRAINT Courses_Employees
    FOREIGN KEY (CourseCoordinatorID)
    REFERENCES Employees (EmployeeID);
```

- Relacja do tabeli `Services`:

```
ALTER TABLE Courses ADD CONSTRAINT Services_Courses
    FOREIGN KEY (ServiceID)
    REFERENCES Services (ServiceID);
```

Relacje odnoszące się do tej tabeli:

- Relacja inicjowana przez tabelę `CourseParticipants`:

```
ALTER TABLE CourseParticipants ADD CONSTRAINT CourseDetails_Courses
    FOREIGN KEY (CourseID)
    REFERENCES Courses (CourseID);
```

- Relacja inicjowana przez tabelę `Modules`:

```
ALTER TABLE Modules ADD CONSTRAINT Modules_Courses
    FOREIGN KEY (CourseID)
    REFERENCES Courses (CourseID);
```

4.3.2 Tabela Modules

Zawiera informacje dotyczące modułów, czyli składowych kursu:

- **ModuleID** int - klucz główny, identyfikator modułu;
- **LanguageID** int - język prowadzenia kursu;
- **CourseID** int - identyfikator kursu, do którego moduł należy;
- **TranslatorID** int nullable - identyfikator tłumacza przypisanego do danego modułu. NULL gdy nie ma tłumacza;
- **ModuleCoordinatorID** int - identyfikator koordynatora modułu;
- **ModuleType** varchar(30) - typ danego modułu kursu.

```
CREATE TABLE Modules (
    ModuleID int  NOT NULL,
    LanguageID int  NOT NULL,
    CourseID int  NOT NULL,
    TranslatorID int  NULL,
    ModuleCoordinatorID int  NOT NULL,
    ModuleType varchar(30) NOT NULL CHECK (ModuleType LIKE '%[a-zA-Z ]%' AND
        LEN(ModuleType) > 0),
    CONSTRAINT Modules_pk PRIMARY KEY  (ModuleID)
);
```

Relacje inicjalizowane przez tą tabelę:

- Relacja do tabeli `Courses`:

```
ALTER TABLE Modules ADD CONSTRAINT Modules_Courses
    FOREIGN KEY (CourseID)
    REFERENCES Courses (CourseID);
```

- Relacja do tabeli `Employees`:

```
ALTER TABLE Modules ADD CONSTRAINT Modules_Employees
    FOREIGN KEY (ModuleCoordinatorID)
    REFERENCES Employees (EmployeeID);
```

- Relacja do tabeli `Languages`:

```
ALTER TABLE Modules ADD CONSTRAINT Modules_Languages
    FOREIGN KEY (LanguageID)
    REFERENCES Languages (LanguageID);
```

- Relacja do tabeli `Translators`:

```
ALTER TABLE Modules ADD CONSTRAINT Modules_Translators
    FOREIGN KEY (TranslatorID)
    REFERENCES Translators (TranslatorID);
```

Relacje odnoszące się do tej tabeli:

- Relacja inicjowana przez tabelę `OfflineVideo`:

```
ALTER TABLE OfflineVideo ADD CONSTRAINT OfflineVideo_Modules
    FOREIGN KEY (ModuleID)
    REFERENCES Modules (ModuleID);
```

- Relacja inicjowana przez tabelę `OnlineLiveMeeting`:

```
ALTER TABLE OnlineLiveMeeting ADD CONSTRAINT OnlineLiveMeeting_Modules
    FOREIGN KEY (ModuleID)
    REFERENCES Modules (ModuleID);
```

- Relacja inicjowana przez tabelę `StationaryMeeting`:

```
ALTER TABLE StationaryMeeting ADD CONSTRAINT StationaryMeeting_Modules
    FOREIGN KEY (ModuleID)
    REFERENCES Modules (ModuleID);
```

4.3.3 Tabela StationaryMeeting

Zawiera informacje na temat spotkań stacjonarnych należących do konkretnego modułu:

- **MeetingID** int - klucz główny, identyfikator spotkania;
- **MeetingDate** datetime - data spotkania;
- **MeetingDuration** time(0) - czas trwania spotkania. Domyślnie 01:30:00, musi być większy niż 00:00:00.
- **ModuleID** int - identyfikator modułu;
- **RoomID** int - identyfikator pomieszczenia, w którym spotkanie ma miejsce;
- **GroupSize** int - liczność grupy, dla której spotkanie jest dostępne. Wielkość grupy musi być dodatnia; – **TeacherID** int - identyfikator prowadzącego spotkanie.

```
CREATE TABLE StationaryMeeting (
    MeetingID int  NOT NULL,
    MeetingDate datetime  NOT NULL,
    MeetingDuration time(0)  NOT NULL DEFAULT '01:30:00' CHECK (MeetingDuration >
        '00:00:00'),
    ModuleID int  NOT NULL,
    RoomID int  NOT NULL,
    GroupSize int  NOT NULL CHECK (GroupSize > 0),
    TeacherID int  NOT NULL,
    CONSTRAINT StationaryMeeting_pk PRIMARY KEY  (MeetingID)
);
```

Relacje inicjalizowane przez tą tabelę:

- Relacja do tabeli `Employees`:

```
ALTER TABLE StationaryMeeting ADD CONSTRAINT StationaryMeeting_Employees
    FOREIGN KEY (TeacherID)
    REFERENCES Employees (EmployeeID);
```

- Relacja do tabeli `Modules`:

```
ALTER TABLE StationaryMeeting ADD CONSTRAINT StationaryMeeting_Modules
    FOREIGN KEY (ModuleID)
    REFERENCES Modules (ModuleID);
```

- Relacja do tabeli `Rooms`:

```
ALTER TABLE StationaryMeeting ADD CONSTRAINT StationaryMeeting_Rooms
    FOREIGN KEY (RoomID)
    REFERENCES Rooms (RoomID);
```

Relacje odnoszące się do tej tabeli:

- Relacja inicjowana przez tabelę `StationaryMeetingDetails`:

```
ALTER TABLE StationaryMeetingDetails ADD CONSTRAINT
    StationaryMeetingDetails_StationaryMeeting
    FOREIGN KEY (MeetingID)
    REFERENCES StationaryMeeting (MeetingID);
```

4.3.4 Tabela StationaryMeetingDetails

Tabela zawierająca informacje o uczestnictwie osób w spotkaniu stacjonarnym:

- **MeetingID** int - część klucza głównego, identyfikator spotkania stacjonarnego;
- **ParticipantID** int - część klucza głównego, identyfikator uczestnika przypisanego do spotkania;
- **Attendance** bit nullable - obecność uczestnika podczas spotkania. NULL jeśli nie jest jeszcze określona.

```
CREATE TABLE StationaryMeetingDetails (
    MeetingID int  NOT NULL,
    ParticipantID int  NOT NULL,
    Attendance bit  NULL,
    CONSTRAINT StationaryMeetingDetails_pk PRIMARY KEY  (MeetingID,ParticipantID)
);
```

Relacje inicjalizowane przez tą tabelę:

- Relacja do tabeli `StationaryMeeting`:

```
ALTER TABLE StationaryMeetingDetails ADD CONSTRAINT
    StationaryMeetingDetails_StationaryMeeting
    FOREIGN KEY (MeetingID)
    REFERENCES StationaryMeeting (MeetingID);
```

- Relacja do tabeli `ServiceUserDetails`:

```
ALTER TABLE StationaryMeetingDetails ADD CONSTRAINT StationaryMeetingDetails_Users
    FOREIGN KEY (ParticipantID)
    REFERENCES ServiceUserDetails (ServiceUserID);
```

4.3.5 Tabela OfflineVideo

Zawiera informacje na temat zajęć realizowanych poprzez nagrania offline:

- **MeetingID** int - klucz główny, identyfikator zajęć;
- **VideoLink** varchar(60) - link do nagrania;
- **ModuleID** int - identyfikator modułu, do którego należy dane nagranie;
- **VideoDuration** time(0) - czas trwania nagrania;
- **TeacherID** int - identyfikator prowadzącego w nagraniu.

```
CREATE TABLE OfflineVideo (
    MeetingID int  NOT NULL,
    VideoLink varchar(60)  NOT NULL,
    ModuleID int  NOT NULL,
    VideoDuration time(0)  NULL DEFAULT '01:30:00' CHECK (VideoDuration > '00:00:00'),
    TeacherID int  NOT NULL,
    CONSTRAINT OfflineVideo_pk PRIMARY KEY  (MeetingID)
);
```

Relacje inicjalizowane przez tą tabelę:

- Relacja do tabeli `Employees`:

```
ALTER TABLE OfflineVideo ADD CONSTRAINT OfflineVideo_Employees
    FOREIGN KEY (TeacherID)
    REFERENCES Employees (EmployeeID);
```

- Relacja do tabeli `Modules`:

```
ALTER TABLE OfflineVideo ADD CONSTRAINT OfflineVideo_Modules
    FOREIGN KEY (ModuleID)
    REFERENCES Modules (ModuleID);
```

Relacje odnoszące się do tej tabeli:

- Relacja inicjowana przez tabelę `OfflineVideoDetails`:

```
ALTER TABLE OfflineVideoDetails ADD CONSTRAINT OfflineVideoDetails_OfflineVideo
    FOREIGN KEY (MeetingID)
    REFERENCES OfflineVideo (MeetingID);
```

4.3.6 Tabela OfflineVideoDetails

Zawiera informacje na temat stanu obejrzenia nagrania przez konkretne osoby:

- **MeetingID** int - klucz główny, identyfikator zajęć offline;
- **ParticipantID** int - identyfikator uczestnika przypisanego do nagrania;
- **dateOfViewing** date nullable - data obejrzenia nagrania. NULL jeśli nagranie nie zostało do tej pory obejrzane.

```
CREATE TABLE OfflineVideoDetails (
    MeetingID int  NOT NULL,
    ParticipantID int  NOT NULL,
    dateOfViewing date  NULL,
    CONSTRAINT OfflineVideoDetails_pk PRIMARY KEY  (MeetingID,ParticipantID)
);
```

Relacje inicjalizowane przez tą tabelę:

- Relacja do tabeli `OfflineVideo`:

```
ALTER TABLE OfflineVideoDetails ADD CONSTRAINT OfflineVideoDetails_OfflineVideo
    FOREIGN KEY (MeetingID)
    REFERENCES OfflineVideo (MeetingID);
```

- Relacja do tabeli `ServiceUserDetails`:

```
ALTER TABLE OfflineVideoDetails ADD CONSTRAINT OfflineVideoDetails_Users
    FOREIGN KEY (ParticipantID)
    REFERENCES ServiceUserDetails (ServiceUserID);
```

4.3.7 Tabela OnlineLiveMeeting

Zawiera informacje na temat zajęć realizowanych jako spotkania na żywo online:

- **MeetingID** int - klucz główny, identyfikator spotkania na żywo;
- **PlatformName** varchar(20) nullable - nazwa platformy na której odbywa się spotkanie. Informacja nieobowiązkowa.
- **Link** varchar(60) nullable - link do spotkania na żywo. NULL jeśli nie został jeszcze wygenerowany;
- **VideoLink** varchar(60) nullable - Link do nagrania ze spotkania na żywo. NULL jeśli nie jest dostępny;
- **ModuleID** int - identyfikator spotkania;
- **MeetingDate** datetime - data spotkania;
- **MeetingDuration** time(0) - czas trwania spotkania. Domyślnie 01:30:00, musi być większy od 00:00:00.

```
CREATE TABLE OnlineLiveMeeting (
    MeetingID int  NOT NULL,
    PlatformName varchar(20)  NULL,
    Link varchar(60)  NULL,
    VideoLink varchar(60)  NULL,
    ModuleID int  NOT NULL,
    MeetingDate datetime  NOT NULL CHECK(MeetingDate > '01-01-2015' AND MeetingDate <
        '01-01-2030'),
    MeetingDuration time(0)  NOT NULL DEFAULT '01:30:00' CHECK (MeetingDuration >
        '00:00:00'),
    TeacherID int  NOT NULL,
    CONSTRAINT OnlineLiveMeeting_pk PRIMARY KEY  (MeetingID)
);
```

Relacje inicjalizowane przez tą tabelę:

- Relacja do tabeli `Employees`:

```
ALTER TABLE OnlineLiveMeeting ADD CONSTRAINT OnlineLiveMeeting_Employees
    FOREIGN KEY (TeacherID)
    REFERENCES Employees (EmployeeID);
```

- Relacja do tabeli `Modules`:

```
ALTER TABLE OnlineLiveMeeting ADD CONSTRAINT OnlineLiveMeeting_Modules
    FOREIGN KEY (ModuleID)
    REFERENCES Modules (ModuleID);
```

Relacje odnoszące się do tej tabeli:

- Relacja inicjowana przez tabelę `OnlineLiveMeetingDetails`:

```
ALTER TABLE OnlineLiveMeetingDetails ADD CONSTRAINT
    OnlineLiveMeetingDetails_OnlineLiveMeeting
    FOREIGN KEY (MeetingID)
    REFERENCES OnlineLiveMeeting (MeetingID);
```

4.3.8 Tabela OnlineLiveMeetingDetails

Zawiera informacje dotyczące uczestnictwa użytkowników w spotkaniu online:

- **MeetingID** int - klucz główny, identyfikator spotkania na żywo online;
- **ParticipantID** int - identyfikator użytkownika przypisanego do uczestnictwa w danym spotkaniu;
- **Attendance** bit nullable - obecność na spotkaniu. NULL jeśli nie jest jeszcze wprowadzona.

```
CREATE TABLE OnlineLiveMeetingDetails (
    MeetingID int  NOT NULL,
    ParticipantID int  NOT NULL,
    Attendance bit  NULL,
    CONSTRAINT OnlineLiveMeetingDetails_pk PRIMARY KEY  (MeetingID,ParticipantID)
);
```

Relacje inicjalizowane przez tą tabelę:

- Relacja do tabeli `OnlineLiveMeeting`:

```
ALTER TABLE OnlineLiveMeetingDetails ADD CONSTRAINT
    OnlineLiveMeetingDetails_OnlineLiveMeeting
    FOREIGN KEY (MeetingID)
    REFERENCES OnlineLiveMeeting (MeetingID);
```

- Relacja do tabeli `ServiceUserDetails`:

```
ALTER TABLE OnlineLiveMeetingDetails ADD CONSTRAINT OnlineLiveMeetingDetails_Users
    FOREIGN KEY (ParticipantID)
    REFERENCES ServiceUserDetails (ServiceUserID);
```

Relacje odnoszące się do tej tabeli:

- Relacja inicjowana przez tabelę `Atonements`:

```
ALTER TABLE Atonements ADD CONSTRAINT Atonements_OnlineLiveMeetingDetails
    FOREIGN KEY (Atoned,StudentID)
    REFERENCES OnlineLiveMeetingDetails (MeetingID,ParticipantID);
```

4.3.9 Tabela CourseParticipants

Tabela par: uczestnik kursu - kurs:

- **ParticipantID** int - część klucza głównego, identyfikator zapisanego na kurs;
- **CourseID** int - część klucza głównego., identyfikator kursu, do którego przypisany jest użytkownik.

```
CREATE TABLE CourseParticipants (
    ParticipantID int  NOT NULL,
    CourseID int  NOT NULL,
    CONSTRAINT CourseDetails_pk PRIMARY KEY  (ParticipantID,CourseID)
);
```

Relacje inicjalizowane przez tą tabelę:

- Relacja do tabeli `Courses`:

```
ALTER TABLE CourseParticipants ADD CONSTRAINT CourseDetails_Courses
    FOREIGN KEY (CourseID)
    REFERENCES Courses (CourseID);
```

- Relacja do tabeli `ServiceUserDetails`:

```
ALTER TABLE CourseParticipants ADD CONSTRAINT CourseDetails_Users
    FOREIGN KEY (ParticipantID)
    REFERENCES ServiceUserDetails (ServiceUserID);
```

4.4 Sekcja systemu opłat

4.4.1 Tabela Payments

Tabela jest wylistowaniem płatności za daną usługę w danym zamówieniu:

- **PaymentID** int - klucz główny, identyfikator płatności;
- **PaymentValue** money - wartość płatności -**PaymentDate** datetime nullable - data płatności, NULL dopóki płatność nie jest zarejestrowana;
- **Service** int - identyfikator usługi;
- **OrderID** int - identyfikator zamówienia.

```
CREATE TABLE Payments (
    PaymentID int  NOT NULL,
    PaymentValue money NOT NULL CHECK (PaymentValue >= 0),
    PaymentDate datetime  NULL,
    ServiceID int  NOT NULL,
    OrderID int  NOT NULL,
    CONSTRAINT Payment_pk PRIMARY KEY  (PaymentID)
);
```

Relacje inicjalizowane przez tą tabelę:

- Relacja do tabeli `OrderDetails`:

```
ALTER TABLE Payments ADD CONSTRAINT Payments_OrderDetails
    FOREIGN KEY (ServiceID,OrderID)
    REFERENCES OrderDetails (ServiceID,OrderID);
```

4.4.2 Tabela OrderDetails

Tabela par postaci zamówienie - usługa. Zawiera informacje na temat usług w konkretnym zamówieniu:

- **OrderID** int - część klucza głównego, identyfikator zamówienia;
- **ServiceID** int - część klucza głównego, identyfikator usługi;
- **PrincipalAgreement** bit - informacja czy dyrektor udzielił zgody na odroczenie płatności.

```
CREATE TABLE OrderDetails (
    OrderID int  NOT NULL,
    ServiceID int  NOT NULL,
    PrincipalAgreement bit NOT NULL
CONSTRAINT OrderDetails_pk PRIMARY KEY  (ServiceID,OrderID)
);
```

Relacje inicjalizowane przez tą tabelę:

- Relacja do tabeli `Orders`:

```
ALTER TABLE OrderDetails ADD CONSTRAINT OrderDetails_Orders
    FOREIGN KEY (OrderID)
    REFERENCES Orders (OrderID);
```

- Relacja do tabeli `Services`:

```
ALTER TABLE OrderDetails ADD CONSTRAINT OrderDetails_Services
    FOREIGN KEY (ServiceID)
    REFERENCES Services (ServiceID);
```

Relacje odnoszące się do tej tabeli:

- Relacja inicjowana przez tabelę `Payments`:

```
ALTER TABLE Payments ADD CONSTRAINT Payments_OrderDetails
    FOREIGN KEY (ServiceID,OrderID)
    REFERENCES OrderDetails (ServiceID,OrderID);
```

4.4.3 Tabela Orders

Zawiera informacje dotyczące daty wykonania i autora zamówienia:

- **OrderID** int - klucz główny, identyfikator zamówienia;
- **UserID** int - właściciel zamówienia;
- **OrderDate** datetime nullable - data złożenia zamówienia. NULL gdy zamówienie nie jest jeszcze zarejestrowane;
- **PaymentLink** varchar(60) nullable - link do systemu zewnętrznego obsługi płatności. NULL gdy nie został jeszcze wygenerowany.

```
CREATE TABLE Orders (
    OrderID int  NOT NULL,
    UserID int  NOT NULL,
    OrderDate datetime CHECK (OrderDate <= GETDATE()),
    PaymentLink varchar(60)  NULL,
    CONSTRAINT Orders_pk PRIMARY KEY  (OrderID)
);
```

Relacje inicjalizowane przez tą tabelę:

- Relacja do tabeli `ServiceUserDetails`:

```
ALTER TABLE Orders ADD CONSTRAINT Orders_Users
    FOREIGN KEY (UserID)
    REFERENCES ServiceUserDetails (ServiceUserID);
```

Relacje odnoszące się do tej tabeli:

- Relacja inicjowana przez tabelę `OrderDetails`:

```
ALTER TABLE OrderDetails ADD CONSTRAINT OrderDetails_Orders
    FOREIGN KEY (OrderID)
    REFERENCES Orders (OrderID);
```

4.4.4 Tabela Services

Stanowi katalog dostępnych dla klienta usług:

- **ServiceID** int - klucz główny, identyfikator usługi;
- **ServiceType** varchar(30) - typ usługi tj. ClassMeeting, Studies, Course, Webinar lub Convention.

```
CREATE TABLE Services (
    ServiceID int  NOT NULL,
    ServiceType varchar(30) NOT NULL CHECK (ServiceType IN ('ClassMeetingService',
        'StudiesService', 'CourseService', 'WebinarService', 'ConventionService')),
    CONSTRAINT Services_pk PRIMARY KEY  (ServiceID)
);
```

Relacje odnoszące się do tej tabeli:

- Relacja inicjowana przez tabelę `ClassMeetingService`:

```
ALTER TABLE ClassMeetingService ADD CONSTRAINT ClassMeetingService_Services
    FOREIGN KEY (ServiceID)
    REFERENCES Services (ServiceID);
```

- Relacja inicjowana przez tabelę `ClassMeeting`:

```
ALTER TABLE ClassMeeting ADD CONSTRAINT ClassMeeting_Services
    FOREIGN KEY (ServiceID)
    REFERENCES Services (ServiceID);
```

- Relacja inicjowana przez tabelę `ConventionService`:

```
ALTER TABLE ConventionService ADD CONSTRAINT ConventionService_Services
    FOREIGN KEY (ServiceID)
    REFERENCES Services (ServiceID);
```

- Relacja inicjowana przez tabelę `Convention`:

```
ALTER TABLE Convention ADD CONSTRAINT Convention_Services
    FOREIGN KEY (ServiceID)
    REFERENCES Services (ServiceID);
```

- Relacja inicjowana przez tabelę `CourseService`:

```
ALTER TABLE CourseService ADD CONSTRAINT CourseService_Services
    FOREIGN KEY (ServiceID)
    REFERENCES Services (ServiceID);
```

- Relacja inicjowana przez tabelę `OrderDetails`:

```
ALTER TABLE OrderDetails ADD CONSTRAINT OrderDetails_Services
    FOREIGN KEY (ServiceID)
    REFERENCES Services (ServiceID);
```

- Relacja inicjowana przez tabelę `Courses`:

```
ALTER TABLE Courses ADD CONSTRAINT Services_Courses
    FOREIGN KEY (ServiceID)
    REFERENCES Services (ServiceID);
```

- Relacja inicjowana przez tabelę `StudiesService`:

```
ALTER TABLE StudiesService ADD CONSTRAINT StudiesService_Services
    FOREIGN KEY (ServiceID)
    REFERENCES Services (ServiceID);
```

- Relacja inicjowana przez tabelę `Studies`:

```
ALTER TABLE Studies ADD CONSTRAINT Studies_Services
    FOREIGN KEY (ServiceID)
    REFERENCES Services (ServiceID);
```

- Relacja inicjowana przez tabelę `WebinarService`:

```
ALTER TABLE WebinarService ADD CONSTRAINT WebinarService_Services
    FOREIGN KEY (ServiceID)
    REFERENCES Services (ServiceID);
```

- Relacja inicjowana przez tabelę `Webinars`:

```
ALTER TABLE Webinars ADD CONSTRAINT Webinars_Services
    FOREIGN KEY (ServiceID)
    REFERENCES Services (ServiceID);
```

4.4.5 Tabela ClassMeetingService

Zawiera informacje o usłudze pojedynczych spotkań studyjnych:

- **ServiceID** int - klucz główny, identyfikator usługi;
- **PriceStudents** money - cena uczestnictwa w spotkaniu dla stałych klientów;
- **PriceOthers** money nullable - cena uczestnictwa dla pozostałych.

```
CREATE TABLE ClassMeetingService (
    ServiceID int  NOT NULL,
    PriceStudents money NOT NULL CHECK (PriceStudents > 0),
    PriceOthers money NULL CHECK (PriceOthers > 0),
    CONSTRAINT ClassMeetingService_pk PRIMARY KEY  (ServiceID)
);
```

Relacje inicjalizowane przez tą tabelę:

- Relacja do tabeli `Services`:

```
ALTER TABLE ClassMeetingService ADD CONSTRAINT ClassMeetingService_Services
    FOREIGN KEY (ServiceID)
    REFERENCES Services (ServiceID);
```

4.4.6 Tabela StudiesService

Tabela informująca o wartości zaliczki dla danych usług studyjnych:

- **ServiceID** int - klucz główny, identyfikator usługi;
- **EntryFee** money - wartość wpisowego za studia.

```
CREATE TABLE StudiesService (
    ServiceID int  NOT NULL,
    EntryFee money NOT NULL CHECK (EntryFee > 0),
    CONSTRAINT StudiesService_pk PRIMARY KEY  (ServiceID)
);
```

Relacje inicjalizowane przez tą tabelę:

- Relacja do tabeli `Services`:

```
ALTER TABLE StudiesService ADD CONSTRAINT StudiesService_Services
    FOREIGN KEY (ServiceID)
    REFERENCES Services (ServiceID);
```

4.4.7 Tabela ConventionService

Zawiera informacje na temat ceny za konkretne zjazdy studyjne:

- **ServiceID** int - klucz główny, identyfikator usługi;
- **Price** money - cena za dany zjazd (tylko dla zapisanych studentów).

```
CREATE TABLE ConventionService (
    ServiceID int  NOT NULL,
    Price money NOT NULL CHECK (Price > 0),
    CONSTRAINT ConventionService_pk PRIMARY KEY  (ServiceID)
);
```

Relacje inicjalizowane przez tą tabelę:

- Relacja do tabeli `Services`:

```
ALTER TABLE ConventionService ADD CONSTRAINT ConventionService_Services
    FOREIGN KEY (ServiceID)
    REFERENCES Services (ServiceID);
```

4.4.8 Tabela CourseService

Tabela zawiera informacje o usłudze kursu - wartość zaliczki i pełna cena:

- **ServiceID** int - klucz główny, identyfikator usługi;
- **AdvanceValue** money - wartość zaliczki na kurs. Musi być niższa, bądź równa wartości całego kursu;
- **FullPrice** money - pełna cena kursu.

```
CREATE TABLE CourseService (
    ServiceID int  NOT NULL,
    AdvanceValue money NOT NULL CHECK (AdvanceValue > 0),
    FullPrice money NOT NULL CHECK (FullPrice >= AdvanceValue),
    CONSTRAINT CourseService_pk PRIMARY KEY  (ServiceID)
);
```

Relacje inicjalizowane przez tą tabelę:

- Relacja do tabeli `Services`:

```
ALTER TABLE CourseService ADD CONSTRAINT CourseService_Services
    FOREIGN KEY (ServiceID)
    REFERENCES Services (ServiceID);
```

4.4.9 Tabela WebinarService

Przechowuje ceny dla konkretnych webinarów:

- **ServiceID** int - klucz główny, identyfikator usługi;
- **Price** money - cena za uczestnictwo w webinarze.

```
CREATE TABLE WebinarService (
    ServiceID int  NOT NULL,
    Price money NOT NULL CHECK (Price > 0),
    CONSTRAINT WebinarService_pk PRIMARY KEY  (ServiceID)
);
```

Relacje inicjalizowane przez tą tabelę:

- Relacja do tabeli `Services`:

```
ALTER TABLE WebinarService ADD CONSTRAINT WebinarService_Services
    FOREIGN KEY (ServiceID)
    REFERENCES Services (ServiceID);
```

4.5 Sekcja pomieszczeń

4.5.1 Tabela Rooms

Tabela zawiera szczegółowe informacje dotyczące dostępnych pomieszczeń:

- **RoomID** int - klucz główny, identyfikator pomieszczenia
- **Capacity** int - liczba miejsc w pomieszczeniu
- **Address** varchar(40) - adres budynku, w którym pomieszczenie się znajduje, domyślnie adres głównego budynku firmy
- **Floor** int nullable - informacja nieobowiązkowa o piętrze;
- **AccesibleForDisabled** bit - informacja o dostępności pomieszczenia dla osób z ograniczeniami dotyczącymi poruszania się, domyślnie pomieszczenia są niedostępne

```
CREATE TABLE Rooms (
    RoomID int  NOT NULL,
    Capacity int NULL CHECK (Capacity > 0),
    Address varchar(40)  NOT NULL DEFAULT 'Kawiory 21, 30-055 Kraków',
    Floor int  NULL CHECK (Floor >= 0),
    AccessibleForDisabled bit  NOT NULL DEFAULT 0,
    CONSTRAINT Rooms_pk PRIMARY KEY  (RoomID)
);
```

Relacje odnoszące się do tej tabeli:

- Relacja inicjowana przez tabelę `RoomDetails`:

```
ALTER TABLE RoomDetails ADD CONSTRAINT RoomDetails_Rooms
    FOREIGN KEY (RoomID)
    REFERENCES Rooms (RoomID);
```

- Relacja inicjowana przez tabelę `StationaryMeeting`:

```
ALTER TABLE StationaryMeeting ADD CONSTRAINT StationaryMeeting_Rooms
    FOREIGN KEY (RoomID)
    REFERENCES Rooms (RoomID);
```

- Relacja inicjowana przez tabelę `StationaryClass`:

```
ALTER TABLE StationaryClass ADD CONSTRAINT StudiesStationaryMeeting_Rooms
    FOREIGN KEY (RoomID)
    REFERENCES Rooms (RoomID);
```

4.5.2 Tabela RoomDetails

Zawiera informacje na temat obłożenia konkretnego pomieszczenia w danym dniu:

- **RoomID** int - część klucza głównego, identyfikator pomieszczenia
- **ScheduleOnDate** date - część klucza głównego, stanowi identyfikator obłożenia w danym dniu
- **SlotID** int - część klucza głównego, identyfikator przedziału czasowego.

```
CREATE TABLE RoomDetails (
    RoomID int  NOT NULL,
    ScheduleOnDate date  NOT NULL,
    SlotID int NOT NULL,
    CONSTRAINT RoomDetails_pk PRIMARY KEY  (RoomID,ScheduleOnDate)
);
```

Relacje inicjalizowane przez tą tabelę:

- Relacja do tabeli `RoomSchedule`:

```
ALTER TABLE RoomDetails ADD CONSTRAINT RoomDetails_RoomScheduleOnDate
    FOREIGN KEY (ScheduleOnDate,RoomID,SlotID)
    REFERENCES RoomSchedule (ScheduleOnDate,RoomID,SlotID);
```

- Relacja do tabeli `Rooms`:

```
ALTER TABLE RoomDetails ADD CONSTRAINT RoomDetails_Rooms
    FOREIGN KEY (RoomID)
    REFERENCES Rooms (RoomID);
```

4.5.3 Tabela RoomSchedule

Tabela przechowuje informacje o dostępności pomieszczenia w konkretnych dniach i przedziałach czasowych.

- **RoomID** int - część klucza głównego, identyfikator pomieszczenia,
- **ScheduleOnDate** date - część klucza głównego, stanowi identyfikator obłożenia w danym dniu
- **SlotID** int - część klucza głównego, identyfikator przedziału czasowego w ciągu dnia
- **StartTime** time - godzina rozpoczęcia przedziału czasowego
- **EndTime** time - godzina zakończenia przedziału czasowego, warunek EndTime > StartTime
- **SlotAvailability** bit - informacja o tym czy sala jest dostępna w danym przedziale czasowym, domyślnie sala jest dostępna

```
CREATE TABLE RoomScheduleOnDate (
    RoomID int NOT NULL,
    ScheduleOnDate date NOT NULL,
    SlotID int NOT NULL,
    StartTime time NOT NULL,
    EndTime time NOT NULL CHECK (EndTime > StartTime),
    SlotAvailability bit NOT NULL DEFAULT 1,
    CONSTRAINT RoomSchedule_pk PRIMARY KEY (RoomID, ScheduleOnDate),
);
```

Relacje odnoszące się do tej tabeli:

- Relacja inicjowana przez tabelę `RoomDetails`:

```
ALTER TABLE RoomDetails ADD CONSTRAINT RoomDetails_RoomScheduleOnDate
    FOREIGN KEY (ScheduleOnDate,RoomID,SlotID)
    REFERENCES RoomSchedule (ScheduleOnDate,RoomID,SlotID);
```

4.6 Sekcja pracowników

4.6.1 Tabela Degrees

Zawiera informacje o stopniach naukowych:

- **DegreeID** int – klucz główny, identyfikator stopnia naukowego
- **DegreeLevel** varchar(30) – poziom stopnia naukowego
- **DegreeName** varchar(30) – nazwa stopnia naukowego

```
CREATE TABLE Degrees (
    DegreeID int NOT NULL,
    DegreeLevel varchar(30) NOT NULL CHECK (LEN(DegreeLevel) > 0),
    DegreeName varchar(30) NOT NULL CHECK (LEN(DegreeName) > 0),
    CONSTRAINT Degrees_pk PRIMARY KEY (DegreeID)
);
```

Relacje odnoszące się do tej tabeli:

- Relacja inicjowana przez tabelę `EmployeeDegree`:

```
ALTER TABLE EmployeeDegree ADD CONSTRAINT DegreeDetails_Degrees
    FOREIGN KEY (DegreeID)
    REFERENCES Degrees (DegreeID);
```

4.6.2 Tabela EmployeeDegree

Zawiera informacje o powiązaniach pracowników z ich stopniami naukowymi:

- **EmployeeID** int – część klucza głównego, identyfikator pracownika
- **DegreeID** int – część klucza głównego, identyfikator stopnia naukowego

```
CREATE TABLE EmployeeDegree (
    EmployeeID int NOT NULL,
    DegreeID int NOT NULL,
    CONSTRAINT DegreeDetails_pk PRIMARY KEY (EmployeeID)
);
```

Relacje inicjalizowane przez tą tabelę:

- Relacja do tabeli `Degrees`:

```
ALTER TABLE EmployeeDegree ADD CONSTRAINT DegreeDetails_Degrees
    FOREIGN KEY (DegreeID)
    REFERENCES Degrees (DegreeID);
```

Relacje odnoszące się do tej tabeli:

- Relacja inicjowana przez tabelę `Employees`:

```
ALTER TABLE Employees ADD CONSTRAINT Employees_DegreeDetails
    FOREIGN KEY (EmployeeID)
    REFERENCES EmployeeDegree (EmployeeID);
```

4.6.3 Tabela Employees

Zawiera informacje o pracownikach:

- **EmployeeID** int – klucz główny, identyfikator pracownika
- **DateOfHire** date nullable – data zatrudnienia

```
CREATE TABLE Employees (
    EmployeeID int NOT NULL,
    DateOfHire date NULL CHECK (DateOfHire <= GETDATE()),
    CONSTRAINT Employees_pk PRIMARY KEY (EmployeeID)
);
```

Relacje inicjalizowane przez tą tabelę:

- Relacja do tabeli `EmployeeDegree`:

```
ALTER TABLE Employees ADD CONSTRAINT Employees_DegreeDetails
    FOREIGN KEY (EmployeeID)
    REFERENCES EmployeeDegree (EmployeeID);
```

- Relacja do tabeli `Users`:

```
ALTER TABLE Employees ADD CONSTRAINT Employees_Users
    FOREIGN KEY (EmployeeID)
    REFERENCES Users (UserID);
```

Relacje odnoszące się do tej tabeli:

- Relacja inicjowana przez tabelę `Courses`:

```
ALTER TABLE Courses ADD CONSTRAINT Courses_Employees
    FOREIGN KEY (CourseCoordinatorID)
    REFERENCES Employees (EmployeeID);
```

- Relacja inicjowana przez tabelę `EmployeesSuperior`:

```
ALTER TABLE EmployeesSuperior ADD CONSTRAINT Employees_EmployeesSuperior
    FOREIGN KEY (EmployeeID)
    REFERENCES Employees (EmployeeID);
```

- Relacja inicjowana przez tabelę `Studies`:

```
ALTER TABLE Studies ADD CONSTRAINT Employees_Studies
    FOREIGN KEY (StudiesCoordinatorID)
    REFERENCES Employees (EmployeeID);
```

- Relacja inicjowana przez tabelę `Translators`:

```
ALTER TABLE Translators ADD CONSTRAINT Employees_Translators
    FOREIGN KEY (TranslatorID)
    REFERENCES Employees (EmployeeID);
```

- Relacja inicjowana przez tabelę `Modules`:

```
ALTER TABLE Modules ADD CONSTRAINT Modules_Employees
    FOREIGN KEY (ModuleCoordinatorID)
    REFERENCES Employees (EmployeeID);
```

- Relacja inicjowana przez tabelę `OfflineVideo`:

```
ALTER TABLE OfflineVideo ADD CONSTRAINT OfflineVideo_Employees
    FOREIGN KEY (TeacherID)
    REFERENCES Employees (EmployeeID);
```

- Relacja inicjowana przez tabelę `OnlineLiveMeeting`:

```
ALTER TABLE OnlineLiveMeeting ADD CONSTRAINT OnlineLiveMeeting_Employees
    FOREIGN KEY (TeacherID)
    REFERENCES Employees (EmployeeID);
```

- Relacja inicjowana przez tabelę `StationaryMeeting`:

```
ALTER TABLE StationaryMeeting ADD CONSTRAINT StationaryMeeting_Employees
    FOREIGN KEY (TeacherID)
    REFERENCES Employees (EmployeeID);
```

- Relacja inicjowana przez tabelę `Webinars`:

```
ALTER TABLE Webinars ADD CONSTRAINT Webinars_Employees
    FOREIGN KEY (TeacherID)
    REFERENCES Employees (EmployeeID);
```

4.6.4 Tabela EmployeesSuperior

Zawiera informacje o przełożonych pracowników:

- **EmployeeID** int – klucz główny, identyfikator pracownika
- **ReportsTo** int nullable – identyfikator przełożonego

```
CREATE TABLE EmployeesSuperior (
    EmployeeID int NOT NULL,
    ReportsTo int NULL,
    CONSTRAINT EmployeesSuperior_pk PRIMARY KEY (EmployeeID)
);
```

Relacje inicjalizowane przez tą tabelę:

- Relacja do tabeli `Employees`:

```
ALTER TABLE EmployeesSuperior ADD CONSTRAINT Employees_EmployeesSuperior
    FOREIGN KEY (EmployeeID)
    REFERENCES Employees (EmployeeID);
```

4.6.5 Tabela Languages

Zawiera informacje o językach:

- **LanguageID** int – klucz główny, identyfikator języka
- **LanguageName** varchar(30) – nazwa języka

```
CREATE TABLE Languages (
    LanguageID int NOT NULL,
    LanguageName varchar(30) NOT NULL CHECK (LEN(LanguageName) > 0),
    CONSTRAINT Languages_pk PRIMARY KEY (LanguageID)
);
```

Relacje odnoszące się do tej tabeli:

- Relacja inicjowana przez tabelę `Modules`:

```
ALTER TABLE Modules ADD CONSTRAINT Modules_Languages
    FOREIGN KEY (LanguageID)
    REFERENCES Languages (LanguageID);
```

- Relacja inicjowana przez tabelę `TranslatorsLanguages`:

```
ALTER TABLE TranslatorsLanguages ADD CONSTRAINT Translators_languages_Languages
    FOREIGN KEY (LanguageID)
    REFERENCES Languages (LanguageID);
```

- Relacja inicjowana przez tabelę `Webinars`:

```
ALTER TABLE Webinars ADD CONSTRAINT Webinars_Languages
    FOREIGN KEY (LanguageID)
    REFERENCES Languages (LanguageID);
```

4.6.6 Tabela Translators

Zawiera informacje o tłumaczach:

- **TranslatorID** int – klucz główny, identyfikator tłumacza

```
CREATE TABLE Translators (
    TranslatorID int NOT NULL,
    CONSTRAINT Translators_pk PRIMARY KEY (TranslatorID)
);
```

Relacje inicjalizowane przez tą tabelę:

- Relacja do tabeli `Employees`:

```
ALTER TABLE Translators ADD CONSTRAINT Employees_Translators
    FOREIGN KEY (TranslatorID)
    REFERENCES Employees (EmployeeID);
```

Relacje odnoszące się do tej tabeli:

- Relacja inicjowana przez tabelę `Modules`:

```
ALTER TABLE Modules ADD CONSTRAINT Modules_Translators
    FOREIGN KEY (TranslatorID)
    REFERENCES Translators (TranslatorID);
```

- Relacja inicjowana przez tabelę `TranslatorsLanguages`:

```
ALTER TABLE TranslatorsLanguages ADD CONSTRAINT Translators_languages_Translators
    FOREIGN KEY (TranslatorID)
    REFERENCES Translators (TranslatorID);
```

- Relacja inicjowana przez tabelę `Webinars`:

```
ALTER TABLE Webinars ADD CONSTRAINT Webinars_Translators
    FOREIGN KEY (TranslatorID)
    REFERENCES Translators (TranslatorID);
```

4.6.7 Tabela TranslatorsLanguages

Zawiera informacje o językach, którymi posługują się tłumacze:

- **TranslatorID** int – klucz główny, identyfikator tłumacza
- **LanguageID** int – klucz główny, identyfikator języka

```
CREATE TABLE TranslatorsLanguages (
    TranslatorID int NOT NULL,
    LanguageID int NOT NULL,
    CONSTRAINT TranslatorsLanguages_pk PRIMARY KEY (LanguageID, TranslatorID)
);
```

Relacje inicjalizowane przez tą tabelę:

- Relacja do tabeli `Languages`:

```
ALTER TABLE TranslatorsLanguages ADD CONSTRAINT Translators_languages_Languages
    FOREIGN KEY (LanguageID)
    REFERENCES Languages (LanguageID);
```

- Relacja do tabeli `Translators`:

```
ALTER TABLE TranslatorsLanguages ADD CONSTRAINT Translators_languages_Translators
    FOREIGN KEY (TranslatorID)
    REFERENCES Translators (TranslatorID);
```

4.7 Sekcja użytkowników

4.7.1 Tabela Locations

Zawiera informacje o lokalizacjach:

- **LocationID** int – klucz główny, identyfikator lokalizacji
- **CountryName** varchar(30) – nazwa kraju
- **ProvinceName** varchar(50) nullable – nazwa województwa
- **CityName** varchar(50) – nazwa miasta

```
CREATE TABLE Locations (
    LocationID int NOT NULL,
    CountryName varchar(30) NOT NULL,
    ProvinceName varchar(50) NULL,
    CityName varchar(50) NOT NULL,
    CONSTRAINT Country_pk PRIMARY KEY (LocationID)
);
```

Relacje odnoszące się do tej tabeli:

- Relacja inicjowana przez tabelę `UserAddressDetails`:

```
ALTER TABLE UserAddressDetails ADD CONSTRAINT Copy_of_Country_StudentsAddressDetails
    FOREIGN KEY (LocationID)
    REFERENCES Locations (LocationID);
```

4.7.2 Tabela ServiceUserDetails

Zawiera informacje o użytkownikach usługi:

- **ServiceUserID** int – klucz główny, identyfikator użytkownika
- **DateOfRegistration** date nullable – data rejestracji, warunek: DateOfRegistration <= GETDATE()

```
CREATE TABLE ServiceUserDetails (
    ServiceUserID int NOT NULL,
    DateOfRegistration date NULL CHECK (DateOfRegistration <= GETDATE()),
    CONSTRAINT Users_pk PRIMARY KEY (ServiceUserID)
);
```

Relacje inicjalizowane przez tą tabelę:

- Relacja do tabeli `Users`:

```
ALTER TABLE ServiceUserDetails ADD CONSTRAINT ServiceUserDetails_Users
    FOREIGN KEY (ServiceUserID)
    REFERENCES Users (UserID);
```

Relacje odnoszące się do tej tabeli:

- Relacja inicjowana przez tabelę `AsyncClassDetails`:

```
ALTER TABLE AsyncClassDetails ADD CONSTRAINT AsyncClassDetails_Users
    FOREIGN KEY (StudentID)
    REFERENCES ServiceUserDetails (ServiceUserID);
```

- Relacja inicjowana przez tabelę `CourseParticipants`:

```
ALTER TABLE CourseParticipants ADD CONSTRAINT CourseDetails_Users
    FOREIGN KEY (ParticipantID)
    REFERENCES ServiceUserDetails (ServiceUserID);
```

- Relacja inicjowana przez tabelę `InternshipDetails`:

```
ALTER TABLE InternshipDetails ADD CONSTRAINT InternshipDetails_Users
    FOREIGN KEY (StudentID)
    REFERENCES ServiceUserDetails (ServiceUserID);
```

- Relacja inicjowana przez tabelę `OfflineVideoDetails`:

```
ALTER TABLE OfflineVideoDetails ADD CONSTRAINT OfflineVideoDetails_Users
    FOREIGN KEY (ParticipantID)
    REFERENCES ServiceUserDetails (ServiceUserID);
```

- Relacja inicjowana przez tabelę `OnlineLiveMeetingDetails`:

```
ALTER TABLE OnlineLiveMeetingDetails ADD CONSTRAINT OnlineLiveMeetingDetails_Users
    FOREIGN KEY (ParticipantID)
    REFERENCES ServiceUserDetails (ServiceUserID);
```

- Relacja inicjowana przez tabelę `Orders`:

```
ALTER TABLE Orders ADD CONSTRAINT Orders_Users
    FOREIGN KEY (UserID)
    REFERENCES ServiceUserDetails (ServiceUserID);
```

- Relacja inicjowana przez tabelę `StationaryMeetingDetails`:

```
ALTER TABLE StationaryMeetingDetails ADD CONSTRAINT StationaryMeetingDetails_Users
    FOREIGN KEY (ParticipantID)
    REFERENCES ServiceUserDetails (ServiceUserID);
```

- Relacja inicjowana przez tabelę `StudiesDetails`:

```
ALTER TABLE StudiesDetails ADD CONSTRAINT StudiesDetails_Users
    FOREIGN KEY (StudentID)
    REFERENCES ServiceUserDetails (ServiceUserID);
```

- Relacja inicjowana przez tabelę `SyncClassDetails`:

```
ALTER TABLE SyncClassDetails ADD CONSTRAINT StudyMeetingDetails_Users
    FOREIGN KEY (StudentID)
    REFERENCES ServiceUserDetails (ServiceUserID);
```

- Relacja inicjowana przez tabelę `SubjectDetails`:

```
ALTER TABLE SubjectDetails ADD CONSTRAINT SubjectDetails_Users
    FOREIGN KEY (StudentID)
    REFERENCES ServiceUserDetails (ServiceUserID);
```

- Relacja inicjowana przez tabelę `WebinarDetails`:

```
ALTER TABLE WebinarDetails ADD CONSTRAINT WebinarDetails_Users
    FOREIGN KEY (ParticipantID)
    REFERENCES ServiceUserDetails (ServiceUserID);
```

4.7.3 Tabela UserAddressDetails

Zawiera informacje o adresach użytkowników:

- **UserID** int – klucz główny, identyfikator użytkownika
- **Address** varchar(30) – adres użytkownika
- **PostalCode** varchar(10) – kod pocztowy
- **LocationID** int – identyfikator lokalizacji

```
CREATE TABLE UserAddressDetails (
    UserID int NOT NULL,
    Address varchar(30) NOT NULL,
    PostalCode varchar(10) NOT NULL,
    LocationID int NOT NULL,
    CONSTRAINT UsersAddressDetails_pk PRIMARY KEY (UserID)
);
```

Relacje inicjalizowane przez tą tabelę:

- Relacja do tabeli `Locations`:

```
ALTER TABLE UserAddressDetails ADD CONSTRAINT Copy_of_Country_StudentsAddressDetails
    FOREIGN KEY (LocationID)
    REFERENCES Locations (LocationID);
```

Relacje odnoszące się do tej tabeli:

- Relacja inicjowana przez tabelę `Users`:

```
ALTER TABLE Users ADD CONSTRAINT Users_UserAddressDetails
    FOREIGN KEY (UserID)
    REFERENCES UserAddressDetails (UserID);
```

4.7.4 Tabela UserContact

Zawiera informacje kontaktowe użytkowników:

- **UserID** int – klucz główny, identyfikator użytkownika
- **Email** varchar(30) – adres e-mail, walidowany, aby zawierał znak "@"
- **Phone** varchar(30) nullable – numer telefonu, wakudiwabt, aby składał sie z 9 cyfr

```
CREATE TABLE UserContact (
    UserID int NOT NULL,
    Email varchar(30) NOT NULL CHECK (Email LIKE '%@%'),
    Phone varchar(30) NULL CHECK (Phone IS NULL OR Phone LIKE '[0-9]{9}'),
    CONSTRAINT UsersContacts_pk PRIMARY KEY (UserID),
    CONSTRAINT EmailUnique_uk UNIQUE (Email)
);
```

Relacje odnoszące się do tej tabeli:

- Relacja inicjowana przez tabelę `Users`:

```
ALTER TABLE Users ADD CONSTRAINT Users_UserContact
    FOREIGN KEY (UserID)
    REFERENCES UserContact (UserID);
```

4.7.5 Tabela UserType

Zawiera informacje o typach użytkowników:

- **UserTypeID** int – klucz główny, identyfikator typu użytkownika
- **UserTypeName** varchar(30) – nazwa typu użytkownika

```
CREATE TABLE UserType (
    UserTypeID int NOT NULL,
    UserTypeName varchar(30) NOT NULL,
    CONSTRAINT EmployeeTypes_pk PRIMARY KEY (UserTypeID)
);
```

Relacje inicjalizowane przez tą tabelę:

- Relacja do tabeli `UserTypePermissionsHierarchy`:

```
ALTER TABLE UserType ADD CONSTRAINT UserType_UserPermissionsHierarchy
    FOREIGN KEY (UserTypeID)
    REFERENCES UserTypePermissionsHierarchy (UserTypeID);
```

Relacje odnoszące się do tej tabeli:

- Relacja inicjowana przez tabelę `Users`:

```
ALTER TABLE Users ADD CONSTRAINT Users_UserType
    FOREIGN KEY (UserTypeID)
    REFERENCES UserType (UserTypeID);
```

4.7.6 Tabela UserTypePermissionsHierarchy

Zawiera informacje o hierarchii uprawnień typów użytkowników:

- **UserTypeID** int – klucz główny, identyfikator typu użytkownika
- **DirectTypeSupervisor** int nullable – identyfikator bezpośredniego przełożonego typu użytkownika, NULL jeśli użytkownik nie ma przełożonego

```
CREATE TABLE UserTypePermissionsHierarchy (
    UserTypeID int NOT NULL,
    DirectTypeSupervisor int NULL,
    CONSTRAINT UserTypePermissionsHierarchy_pk PRIMARY KEY (UserTypeID)
);
```

Relacje odnoszące się do tej tabeli:

- Relacja inicjowana przez tabelę `UserType`:

```
ALTER TABLE UserType ADD CONSTRAINT UserType_UserPermissionsHierarchy
    FOREIGN KEY (UserTypeID)
    REFERENCES UserTypePermissionsHierarchy (UserTypeID);
```

4.7.7 Tabela Users

Zawiera informacje o użytkownikach:

- **UserID** int – klucz główny, identyfikator użytkownika
- **FirstName** varchar(30) – imię użytkownika
- **LastName** varchar(30) – nazwisko użytkownika
- **DateOfBirth** date nullable – data urodzenia, warunek: DateOfBirth <= GETDATE()
- **UserTypeID** int – identyfikator typu użytkownika

```
CREATE TABLE Users (
    UserID int NOT NULL,
    FirstName varchar(30) NOT NULL,
    LastName varchar(30) NOT NULL,
    DateOfBirth date NULL CHECK (DateOfBirth <= GETDATE()),
    UserTypeID int NOT NULL,
    CONSTRAINT Users_pk PRIMARY KEY (UserID)
);

#### Relacje inicjalizowane przez tą tabelę:
- Relacja do tabeli `UserAddressDetails`:
```sql
ALTER TABLE Users ADD CONSTRAINT Users_UserAddressDetails
    FOREIGN KEY (UserID)
    REFERENCES UserAddressDetails (UserID);
```

- Relacja do tabeli `UserContact`:

```
ALTER TABLE Users ADD CONSTRAINT Users_UserContact
    FOREIGN KEY (UserID)
    REFERENCES UserContact (UserID);
```

- Relacja do tabeli `UserType`:

```
ALTER TABLE Users ADD CONSTRAINT Users_UserType
    FOREIGN KEY (UserTypeID)
    REFERENCES UserType (UserTypeID);
```
```

Relacje odnoszące się do tej tabeli:

- Relacja inicjowana przez tabelę `Employees`:

```
ALTER TABLE Employees ADD CONSTRAINT Employees_Users
    FOREIGN KEY (EmployeeID)
    REFERENCES Users (UserID);
```

- Relacja inicjowana przez tabelę `ServiceUserDetails`:

```
ALTER TABLE ServiceUserDetails ADD CONSTRAINT ServiceUserDetails_Users
    FOREIGN KEY (ServiceUserID)
    REFERENCES Users (UserID);
```

5 Widoki

5.0.1 Widok ATTENDANCE_LISTS_COURSES - Emil Żychowicz

Widok przedstawia informacje o obecności konkretnych uczestników na zajęciach należących do kursów:

- **CourseName** - nazwa kursu,
- **CourseID** - identyfikator kursu,
- **ModuleID** - indeks modułu do którego należy dane spotkanie,
- **MeetingID** - identyfikator spotkania,
- **PariticipantID** - data spotkania,
- **MeetingType** - typ spotkania,
- **FirstName** - imię uczestnika,
- **LastName** - nazwisko uczestnika,
- **Attendance** - wartość bitowa czy był obecny,
- **MeetingDate** - data spotkania.

```
create view ATTENDANCE_LISTS_COURSES as
select
  c.CourseName,
  c.CourseID,
  m.ModuleID,
  smd.MeetingID,
  'StationaryClass' as MeetingType,
  smd.ParticipantID,
  u.FirstName,
  u.LastName,
  smd.Attendance,
  sm.MeetingDate
from StationaryMeetingDetails as smd
INNER JOIN StationaryMeeting as sm
  on sm.MeetingID = smd.MeetingID
INNER JOIN Modules as m
  on sm.ModuleID = m.ModuleID
INNER JOIN Courses as c
  on c.CourseID = m.CourseID
INNER JOIN ServiceUserDetails as us
  on us.ServiceUserID = smd.ParticipantID
INNER JOIN Users as u
  on u.UserID = us.ServiceUserID
  where sm.MeetingDate < CONVERT(DATE, GETDATE())
union
select
  c.CourseName,
  c.CourseID,
  m.ModuleID,
  omd.MeetingID,
```

```
  'OnlineLive' as MeetingType,
  omd.ParticipantID,
  u.FirstName,
  u.LastName,
  omd.Attendance,
  om.MeetingDate
from OnlineLiveMeetingDetails as omd
INNER JOIN OnlineLiveMeeting as om
  on om.MeetingID = omd.MeetingID
INNER JOIN Modules as m
  on om.ModuleID = m.ModuleID
INNER JOIN Courses as c
  on c.CourseID = m.CourseID
INNER JOIN ServiceUserDetails as us
  on us.ServiceUserID = omd.ParticipantID
INNER JOIN Users as u
  on u.UserID = us.ServiceUserID
  where om.MeetingDate < CONVERT(DATE, GETDATE())
```

5.0.2 Widok ATTENDANCE_LISTS_OFFLINEVIDEO_COURSES - Emil Żychowicz

Widok przedstawia listę uczestników kursów, którzy brali udział w spotkaniach typu "Offline Video". Dla każdego uczestnika i spotkania, wyświetlane są szczegóły dotyczące obecności na spotkaniach:

- **CourseName** – nazwa kursu, do którego należy spotkanie.
- **CourseID** – identyfikator kursu.
- **ModuleID** – identyfikator modułu, do którego należy spotkanie.
- **MeetingID** – identyfikator spotkania.
- **MeetingType** – typ spotkania: "OfflineVideo"(spotkanie offline).
- **ParticipantID** – identyfikator uczestnika spotkania.
- **FirstName** – imię uczestnika.
- **LastName** – nazwisko uczestnika.
- **Attendance** – obecność na spotkaniu (0 oznacza brak obecności, 1 oznacza obecność).
- **dateOfViewing** – data oglądania spotkania (jeśli jest dostępna). Widok obejmuje wyłącznie spotkania, dla których data oglądania nie została jeszcze zarejestrowana lub data oglądania jest wcześniejsza niż bieżąca data.

```
create view ATTENDANCE_LISTS_OFFLINEVIDEO_COURSES as
select
  c.CourseName,
  c.CourseID,
  m.ModuleID,
  omd.MeetingID,
  'OfflineVideo' as MeetingType,
  omd.ParticipantID,
  u.FirstName,
  u.LastName,
  CASE WHEN (omd.dateOfViewing) IS NULL THEN 0 ELSE 1 END AS Attendance,
  omd.dateOfViewing
from OfflineVideoDetails as omd
INNER JOIN OfflineVideo as om
  on om.MeetingID = omd.MeetingID
INNER JOIN Modules as m
  on om.ModuleID = m.ModuleID
INNER JOIN Courses as c
  on c.CourseID = m.CourseID
INNER JOIN ServiceUserDetails as us
  on us.ServiceUserID = omd.ParticipantID
INNER JOIN Users as u
  on u.UserID = us.ServiceUserID
  where omd.dateOfViewing IS  NULL OR omd.dateOfViewing < CONVERT(DATE, GETDATE())
```

5.0.3 Widok ATTENDANCE_MEETINGS_IN_COURSES - Emil Żychowicz

Widok przedstawia statystyki obecności na spotkaniach w ramach kursów, z podziałem na typ spotkania (stacjonarne i online):

- **CourseName** – nazwa kursu, do którego należy spotkanie.
- **CourseID** – identyfikator kursu.
- **ModuleID** – identyfikator modułu, do którego należy spotkanie.
- **MeetingID** – identyfikator spotkania.
- **MeetingDate** – data spotkania.
- **MeetingType** – typ spotkania: "Stationary"(stacjonarne) lub "Online"(zdalne).
- **Attendance** – procent obecności.

Widok obejmuje wyłącznie spotkania, które odbyły się przed bieżącą datą.

```
create view ATTENDANCE_MEETINGS_IN_COURSES as
select
  c.CourseName, c.CourseID, m.ModuleID, smd.MeetingID, sm.MeetingDate, 'Stationary' as
      MeetingType,
   ROUND(CAST((SELECT COUNT(smd1.ParticipantID)
         FROM StationaryMeetingDetails AS smd1
         WHERE smd1.MeetingID = smd.MeetingID AND smd1.Attendance = 1) AS FLOAT) /
             CAST(COUNT(smd.ParticipantID) AS FLOAT),2) AS Attendance
from StationaryMeetingDetails as smd
INNER JOIN StationaryMeeting as sm
  on sm.MeetingID = smd.MeetingID
INNER JOIN Modules as m
  on sm.ModuleID = m.ModuleID
INNER JOIN Courses as c
  on c.CourseID = m.CourseID
group by smd.MeetingID, sm.MeetingDate, c.CourseID, m.ModuleID, c.CourseName
having sm.MeetingDate < CONVERT(DATE, GETDATE())
union
select
  c.CourseName, c.CourseID, m.ModuleID, omd.MeetingID, om.MeetingDate, 'OnlineLive' as
      MeetingType,
   ROUND(CAST((SELECT COUNT(omd1.ParticipantID)
         FROM OnlineLiveMeetingDetails AS omd1
         WHERE omd1.MeetingID = omd.MeetingID AND omd1.Attendance = 1) AS FLOAT) /
             CAST(COUNT(omd.ParticipantID) AS FLOAT),2) AS Attendance
from OnlineLiveMeetingDetails as omd
INNER JOIN OnlineLiveMeeting as om
  on om.MeetingID = omd.MeetingID
INNER JOIN Modules as m
  on om.ModuleID = m.ModuleID
INNER JOIN Courses as c
  on c.CourseID = m.CourseID
group by omd.MeetingID, om.MeetingDate, c.CourseID, m.ModuleID, c.CourseName
having om.MeetingDate < CONVERT(DATE, GETDATE())
```

5.0.4 Widok CLASSMEETING_USERS - Emil Żychowicz

Widok przedstawia informacje o użytkownikach przypisanych do spotkań w ramach zajęć:

- **ClassMeetingID** – identyfikator spotkania w ramach zajęć.
- **ServiceID** – identyfikator usługi związanej ze spotkaniem.
- **SubjectID** – identyfikator przedmiotu powiązanego z zajęciami.
- **MeetingName** – nazwa spotkania.
- **MeetingType** – typ spotkania.
- **ServiceUserID** – identyfikator uczestnika.
- **FirstName** – imię uczestnika.
- **LastName** – nazwisko uczestnika.
- **SyncAttendance** – informacja o obecności w przypadku zajęć synchronicznych.
- **AsyncViewDate** – data obejrzenia materiałów w przypadku zajęć asynchronicznych.

```sql
CREATE VIEW CLASSMEETING_USERS AS
SELECT
    cm.ClassMeetingID,
  cm.ServiceID,
    cm.SubjectID,
    cm.MeetingName,
    cm.MeetingType,
    st.ServiceUserID,
    u.FirstName,
    u.LastName,
    scd.Attendance AS SyncAttendance,
    acd.ViewDate AS AsyncViewDate
FROM ClassMeeting cm
LEFT JOIN SyncClassDetails scd
    ON scd.MeetingID = cm.ClassMeetingID
LEFT JOIN AsyncClassDetails acd
    ON acd.MeetingID = cm.ClassMeetingID

LEFT JOIN ServiceUserDetails st
    ON st.ServiceUserID = scd.StudentID
       OR st.ServiceUserID = acd.StudentID

LEFT JOIN Users u
    ON u.UserID = st.ServiceUserID
WHERE scd.StudentID IS NOT NULL
      OR acd.StudentID IS NOT NULL
;
```

5.0.5 Widok CONVENTION_INCOME - Emil Żychowicz

Widok przedstawia dane o przychodach związanych z zjazdami:

- **ConventionID** – identyfikator zjazdu.
- **Income** – wartość przychodów związanych ze zjazdem.

```sql
CREATE VIEW CONVENTION_INCOME as
SELECT c.ConventionID, si.Income
FROM Convention as c
INNER JOIN SERVICE_ID_INCOME AS si
    ON c.ServiceID = si.ServiceID
```

5.0.6 Widok CONVENTION_USERS - Emil Żychowicz

Widok zawiera informacje o użytkownikach przypisanych do zjazdu w danym semestrze:

- **ConventionID** – identyfikator konwencji.
- **ServiceID** – identyfikator usługi powiązanej ze zjazdem.
- **SemesterID** – identyfikator semestru, w którym odbywa się zjazd.
- **SubjectID** – identyfikator przedmiotu powiązanego ze zjazdem.
- **SubjectName** – nazwa przedmiotu.
- **ServiceUserID** – identyfikator uczestnika.
- **FirstName** – imię uczestnika.
- **LastName** – nazwisko uczestnika.

Widok łączy dane z informacjami o przedmiocie, semestrze i użytkownikach biorących udział w zjeździe.

```sql
CREATE VIEW CONVENTION_USERS AS
SELECT
    c.ConventionID,
  c.ServiceID,
    c.SemesterID,
```

```
    c.SubjectID,
    sub.SubjectName,
    st.ServiceUserID,
  u.FirstName,
    u.LastName
FROM Convention c
JOIN SemesterDetails sem
    ON sem.SemesterID = c.SemesterID
JOIN StudiesDetails sd
    ON sd.StudiesID = sem.StudiesID
JOIN ServiceUserDetails st
    ON st.ServiceUserID = sd.StudentID
JOIN Users u
    ON u.UserID = st.ServiceUserID
JOIN SubjectStudiesAssignment s2s
    ON s2s.StudiesID = sem.StudiesID
    AND s2s.SubjectID = c.SubjectID
JOIN Subject sub
    ON sub.SubjectID = c.SubjectID
;
```

5.0.7 Widok COURSES_USERS - Jakub Kaliński

Widok przedstawia informacje o użytkownikach zapisanych na kursy:

- **CourseID** – identyfikator kursu.
- **ServiceID** – identyfikator usługi powiązanej z kursem.
- **CourseName** – nazwa kursu.
- **ServiceUserID** – identyfikator użytkownika powiązanego z kursem.
- **FirstName** – imię użytkownika.
- **LastName** – nazwisko użytkownika.

```
create view COURSES_USERS as
select cp.CourseID, c.ServiceID, c.CourseName, us.ServiceUserID, u.FirstName, u.LastName
  from CourseParticipants as cp
  INNER JOIN ServiceUserDetails as us
  on cp.ParticipantID = us.ServiceUserID
  INNER JOIN Courses as c
  on c.CourseID = cp.CourseID
  INNER JOIN Users as u
  on u.UserID = us.ServiceUserID
```

5.0.8 Widok COURSE_INCOME - Emil Żychowicz

Widok prezentuje dane o przychodach związanych z kursami:

- **CourseID** – identyfikator kursu.
- **CourseName** – nazwa kursu.
- **Income** – suma przychodów powiązanych z usługą danego kursu.

```
CREATE VIEW COURSE_INCOME as
SELECT c.CourseID, c.CourseName, si.Income
FROM Courses AS c
INNER JOIN SERVICE_ID_INCOME AS si
    ON c.ServiceID = si.ServiceID
```

5.0.9 Widok FINANCIAL_REPORT - Emil Żychowicz

Widok przedstawia raport finansowy z podziałem na typy usług:

- **Service Type** – typ usługi, np. kurs, spotkanie, konwencja.

- **Income** – suma przychodów wygenerowanych przez dany typ usługi.

Widok uwzględnia wyłącznie płatności, które zostały dokonane (mają datę płatności).

```
create view FINANCIAL_REPORT as
select s.ServiceType as 'Service Type' , sum(p.PaymentValue) as 'Income'
  from Payments as p
  INNER JOIN OrderDetails as od
  on od.OrderID = p.OrderID AND od.ServiceID = p.ServiceID
  INNER JOIN Services as s
  on s.ServiceID = p.ServiceID
  where p.PaymentDate IS NOT NULL
  group by s.ServiceType
```

5.0.10 Widok FULL_PRICE - Emil Żychowicz

Widok przedstawia pełne ceny różnych rodzajów usług. Nie uwzględnia promocji dla studentów na pojedyncze spotkania:

- **ServiceID** – identyfikator usługi.
- **FullPrice** – pełna cena usługi, różniąca się w zależności od jej rodzaju (np. kurs, spotkanie, webinar).

```
create view FULL_PRICE as
select T1.ServiceID, T1.FullPrice
  from
    (select ServiceID, PriceOthers as FullPrice from ClassMeetingService
    union
    select ServiceID, EntryFee as FullPrice from StudiesService
    union
    select ServiceID, Price as FullPrice from ConventionService
    union
    select ServiceID, FullPrice from CourseService
    union
    select ServiceID, Price from WebinarService
    ) as T1
```

5.0.11 Widok IN_DEBT_USERS - Jakub Kaliński

Widok przedstawia użytkowników, którzy mają zaległości finansowe za zakończone usługi:

- **UserID** – identyfikator użytkownika.
- **FirstName** – imię użytkownika.
- **LastName** – nazwisko użytkownika.
- **InDebt** – kwota zaległości (suma opłat do zapłaty minus wpłacona kwota).
- **ServiceType** – typ usługi, za którą użytkownik ma zaległości.

```
CREATE VIEW IN_DEBT_USERS AS

-- CTE z sumą płatności za zakonczone uslugi dla uzytkownikow ktorzy korzystali z uslug
    zakonczonych
WITH Suspects AS (
    SELECT
        o.UserID,
        p.OrderID,
        p.ServiceID,
        SUM(p.PaymentValue) AS Paid
    FROM Orders AS o
    LEFT OUTER JOIN OrderDetails AS od
        ON o.OrderID = od.OrderID
    LEFT OUTER JOIN Payments AS p
        ON p.OrderID = od.OrderID AND p.ServiceID = od.ServiceID AND
            od.PrincipalAgreement = 0 --jesli ma zgode dyrektora na odroczenie to nie
            licz danej uslugi dla danego zamowienia
```

```
    WHERE p.PaymentDate IS NOT NULL AND o.UserID IN (SELECT ServiceUserID FROM
        SERVICE_USERS WHERE ServiceID = p.ServiceID)
      AND p.ServiceID IN (
          SELECT s_e_services.ServiceID
          FROM START_END_OF_SERVICES AS s_e_services
          WHERE s_e_services.EndOfService < CONVERT(DATE, GETDATE())
      )
    GROUP BY o.UserID, p.OrderID, p.ServiceID
)

SELECT
    Suspects.UserID,
  (SELECT u.FirstName  FROM Users AS u WHERE Suspects.UserID = u.UserID) as FirstName,
  (SELECT u.LastName  FROM Users AS u WHERE Suspects.UserID = u.UserID) as LastName,
    Suspects.Paid - toPay AS InDebt,
    (SELECT s.ServiceType FROM Services as s where Suspects.ServiceID = s.ServiceID) as
        ServiceType
FROM Suspects
CROSS APPLY ( --dla kazdego wiersza stosuje ponizsze obliczenia
    SELECT
        CASE
            WHEN Suspects.UserID IN (SELECT StudentID FROM StudiesDetails)
                 AND Suspects.ServiceID IN (SELECT ServiceUserID FROM CLASSMEETING_USERS)
            THEN (SELECT cms.PriceStudents
                  FROM ClassMeetingService AS cms
                  WHERE cms.ServiceID = Suspects.ServiceID)
            ELSE (SELECT fp.FullPrice FROM FULL_PRICE AS fp where fp.ServiceID =
                Suspects.ServiceID)
        END AS toPay
) AS Calculations
WHERE Suspects.Paid < Calculations.toPay;
```

5.0.12 Widok PARTICIPANTS_MEETINGS_FUTURE_COURSES - Emil Żychowicz

Widok przedstawia listę uczestników, którzy są zapisani na przyszłe spotkania w ramach kursów, podzielone na spotkania stacjonarne oraz online:

- **MeetingID** – identyfikator spotkania.
- **MeetingDate** – data spotkania.
- **MeetingType** – typ spotkania: "Stationary"(stacjonarne) lub "Online"(zdalne).
- **ModuleID** – identyfikator modułu, do którego należy spotkanie.
- **CourseID** – identyfikator kursu, do którego należy spotkanie.
- **CourseName** – nazwa kursu.
- **ParticipantID** – identyfikator uczestnika spotkania.
- **FirstName** – imię uczestnika.
- **LastName** – nazwisko uczestnika.

Widok obejmuje wyłącznie spotkania, które odbywają się w przyszłości.

```
create view PARTICIPANTS_MEETINGS_FUTURE_COURSES as
select
  smd.MeetingID,
  sm.MeetingDate,
  'Stationary' as MeetingType,
  m.ModuleID,
  c.CourseID,
  c.CourseName,
  smd.ParticipantID,
  u.FirstName,
  u.LastName
from StationaryMeetingDetails as smd
INNER JOIN StationaryMeeting as sm
  on sm.MeetingID = smd.MeetingID
INNER JOIN Modules as m
```

```
  on sm.ModuleID = m.ModuleID
INNER JOIN Courses as c
  on c.CourseID = m.CourseID
INNER JOIN ServiceUserDetails as us
  on us.ServiceUserID = smd.ParticipantID
INNER JOIN Users as u
  on u.UserID = us.ServiceUserID
where sm.MeetingDate > CONVERT(DATE, GETDATE())
union
select
  omd.MeetingID,
  om.MeetingDate,
  'Online' as MeetingType,
  m.ModuleID,
  c.CourseID,
  c.CourseName,
  omd.ParticipantID,
  u.FirstName,
  u.LastName
from OnlineLiveMeetingDetails as omd
INNER JOIN OnlineLiveMeeting as om
  on om.MeetingID = omd.MeetingID
INNER JOIN Modules as m
  on om.ModuleID = m.ModuleID
INNER JOIN Courses as c
  on c.CourseID = m.CourseID
INNER JOIN ServiceUserDetails as us
  on us.ServiceUserID = omd.ParticipantID
INNER JOIN Users as u
  on u.UserID = us.ServiceUserID
where om.MeetingDate > CONVERT(DATE, GETDATE())
```

5.0.13 Widok NUM_OF_PARTICIPANTS_FUTURE_COURSES - Emil Żychowicz

Widok zawiera informacje o liczbie uczestników zapisanych na przyszłe kursy:

- **CourseID** – identyfikator kursu.
- **CourseName** – nazwa kursu.
- **Number of participants** – liczba uczestników zapisanych na kurs.

Widok uwzględnia wyłącznie kursy, które odbędą się w przyszłości.

```
create view NUM_OF_PARTICIPANTS_FUTURE_COURSES as
select c.CourseID, c.CourseName, COUNT(cp.ParticipantID) as 'Number of participants'
from CourseParticipants as cp
INNER JOIN Courses as c
on c.CourseID = cp.CourseID
group by c.CourseID, c.CourseName, c.CourseDate
having c.CourseDate > CONVERT(DATE, GETDATE())
```

5.0.14 Widok NUM_OF_PARTICIPANTS_MEETINGS_COURSES - Emil Żychowicz

Widok prezentuje liczbę uczestników biorących udział w przyszłych spotkaniach kursów, z podziałem na typy spotkań:

- **CourseID** – identyfikator kursu.
- **ModuleID** – identyfikator modułu, do którego należy spotkanie.
- **MeetingID** – identyfikator spotkania.
- **MeetingDate** – data spotkania.
- **MeetingType** – typ spotkania: "Stationary"(stacjonarne) lub "Online"(zdalne).
- **Number of Participants** – liczba uczestników zapisanych na spotkanie.

```
create or alter view NUM_OF_PARTICIPANTS_MEETINGS_COURSES as
select m.CourseID, sm.ModuleID, smd.MeetingID, sm.MeetingDate, 'Stationary' as
    MeetingType, COUNT(smd.ParticipantID) as [Number of Participants]
  from StationaryMeetingDetails as smd
  INNER JOIN StationaryMeeting as sm
  on sm.MeetingID = smd.MeetingID
  INNER JOIN Modules as m
  on sm.ModuleID = m.ModuleID
  group by smd.MeetingID, sm.MeetingDate,m.CourseID, sm.ModuleID
  having sm.MeetingDate > CONVERT(DATE, GETDATE())
union
select m.CourseID, om.ModuleID, omd.MeetingID, om.MeetingDate, 'Online' as MeetingType,
    COUNT(omd.ParticipantID) as [Number of Participants]
  from OnlineLiveMeetingDetails as omd
  INNER JOIN OnlineLiveMeeting as om
  on om.MeetingID = omd.MeetingID
  INNER JOIN Modules as m
  on om.ModuleID = m.ModuleID
  group by omd.MeetingID, om.MeetingDate,m.CourseID, om.ModuleID
  having om.MeetingDate > CONVERT(DATE, GETDATE())
```

5.0.15 Widok SERVICE_ID_INCOME - Emil Żychowicz

Widok przedstawia dane o przychodach z podziałem na usługi:

- **ServiceID** – identyfikator usługi.
- **Income** – suma przychodów dla danej usługi.
- **ServiceType** - typ usługi.

```
create or alter view SERVICE_ID_INCOME as
select p.ServiceID, s.ServiceType, sum(p.PaymentValue) as 'Income'
  from Payments as p
  INNER JOIN OrderDetails as od
  on od.OrderID = p.OrderID AND od.ServiceID = p.ServiceID
  INNER JOIN Services as s
  on s.ServiceID = od.ServiceID
  group by p.ServiceID, s.ServiceType
```

5.0.16 Widok SERVICE_USERS - Jakub Kaliński

Widok łączy użytkowników z usługami, w których biorą udział, niezależnie od ich rodzaju:

- **ServiceID** – identyfikator usługi.
- **ServiceUserID** – identyfikator użytkownika przypisanego do usługi.

Widok obejmuje dane dla usług takich jak studia, kursy, spotkania, konwencje i webinary.

```
create view SERVICE_USERS as
select ServiceID, ServiceUserID from STUDIES_USERS
union
select ServiceID, ServiceUserID from COURSES_USERS
union
select ServiceID, ServiceUserID from CLASSMEETING_USERS
union
select ServiceID, ServiceUserID from CONVENTION_USERS
union
select ServiceID, ServiceUserID from WEBINAR_USERS
```

5.0.17 Widok START_END_OF_CLASSMEETING - Emil Żychowicz

Widok prezentuje informacje o datach rozpoczęcia i zakończenia spotkań w ramach zajęć:

- **ServiceID** – identyfikator usługi związanej ze spotkaniem.
- **StartOfService** – data rozpoczęcia spotkania.
- **EndOfService** – data zakończenia spotkania.

Widok uwzględnia różne rodzaje spotkań, takie jak stacjonarne, online czy wideo offline.

```
create view START_END_OF_CLASSMEETING as
select CM.ServiceID, T1.StartDate as StartOfService, T1.EndDate as EndOfService
  from ClassMeeting as CM
  INNER JOIN (
    select MeetingID, CAST(StartDate as DATE) as StartDate,CAST(DATEADD(SECOND,
            DATEDIFF(SECOND, '00:00:00', Duration),
            StartDate) as DATE)  as EndDate from OnlineLiveClass
    UNION
    select MeetingID, CAST(StartDate as DATE) as StartDate, CAST(DATEADD(SECOND,
            DATEDIFF(SECOND, '00:00:00', Duration),
            StartDate) as DATE) as EndDate from StationaryClass
    UNION
    select MeetingID, StartDate, Deadline as EndDate from OfflineVideoClass
  ) as T1
  on T1.MeetingID = CM.ClassMeetingID
  group by CM.ServiceID, T1.StartDate, T1.EndDate
```

5.0.18 Widok START_END_OF_CONVENTION - Emil Żychowicz

Widok przedstawia daty rozpoczęcia i zakończenia konwencji:

- **ServiceID** – identyfikator usługi powiązanej z konwencją.
- **StartOfService** – data rozpoczęcia konwencji.
- **EndOfService** – data zakończenia konwencji, obliczona jako suma daty rozpoczęcia i czasu trwania.

```
create view START_END_OF_CONVENTION as
select c.ServiceID, c.StartDate as StartOfService, DATEADD(DAY, c.Duration, c.StartDate)
    AS EndOfService
  from Convention as c
```

5.0.19 Widok START_END_OF_COURSES - Emil Żychowicz

Widok przedstawia daty rozpoczęcia i zakończenia kursów:

- **ServiceID** – identyfikator usługi powiązanej z kursem.
- **StartOfService** – data rozpoczęcia kursu.
- **EndOfService** – data zakończenia kursu, wyznaczona jako ostatnia data spotkania powiązanego z kursem.

```
create view START_END_OF_COURSES as
select c.ServiceID, c.CourseDate as StartOfService, CAST(max(T1.MeetingDate) AS DATE) as
    EndOfService
  from Courses as c
  INNER JOIN Modules as m
  on c.CourseID = m.CourseID
  INNER JOIN (
    select MeetingDate, ModuleID from OnlineLiveMeeting
    UNION
    select MeetingDate, ModuleID from StationaryMeeting) as T1
  on T1.ModuleID = m.ModuleID
  group by c.ServiceID, c.CourseDate
```

5.0.20 Widok START_END_OF_SERVICES - Emil Żychowicz

Widok łączy dane o datach rozpoczęcia i zakończenia różnych usług:

- **ServiceID** – identyfikator usługi.
- **StartOfService** – data rozpoczęcia usługi.
- **EndOfService** – data zakończenia usługi.

Dane pochodzą z widoków: **START_END_OF_CONVENTION**, **START_END_STUDIES**, **START_END_OF_COURS** **START_END_OF_WEBINAR** oraz **START_END_OF_CLASSMEETING**.

```
create view START_END_OF_SERVICES as
select ServiceID, StartOfService, EndOfService
  from START_END_OF_CONVENTION
union
select ServiceID, StartOfService, EndOfService
  from START_END_STUDIES
union
select ServiceID, StartOfService, EndOfService
  from START_END_OF_COURSES
union
select ServiceID, StartOfService, EndOfService
  from START_END_OF_CLASSMEETING
union
select ServiceID, StartOfService, EndOfService
  from START_END_OF_WEBINAR
```

5.0.21 Widok START_END_STUDIES - Emil Żychowicz

Widok przedstawia daty rozpoczęcia i zakończenia studiów:

- **ServiceID** – identyfikator usługi związanej ze studiami.
- **StartOfService** – data rozpoczęcia studiów (termin zapisów).
- **EndOfService** – data zakończenia studiów (przewidywana data ukończenia).

```
create view START_END_STUDIES as
select ServiceID, CAST(EnrollmentDeadline as DATE) as StartOfService,
    CAST(ExpectedGraduationDate as DATE) as EndOfService
  from Studies
```

5.0.22 Widok START_END_OF_WEBINAR - Emil Żychowicz

Widok **START_END_OF_WEBINAR** oblicza daty rozpoczęcia i zakończenia webinarów na podstawie daty rozpoczęcia oraz czasu trwania webinaru zawartego w tabeli **Webinars**.

- **ServiceID** – identyfikator usługi powiązanej z danym webinarem.
- **StartOfService** – data rozpoczęcia webinaru. Jest to wartość **WebinarDate**, przekształcona na format **DATE**, co oznacza, że będzie zawierała tylko datę bez czasu.
- **EndOfService** – data zakończenia webinaru. Jest obliczana przez dodanie czasu trwania (**DurationTime**) do **WebinarDate** i konwersję wyniku na format **DATE** (data, bez godziny, minut i sekund).

```
create view START_END_OF_WEBINAR as
select
    w.ServiceID,
    CAST(T1.WebinarDate as DATE) as StartOfService,
    CAST(DATEADD(SECOND, DATEDIFF(SECOND, '00:00:00', T1.DurationTime), T1.WebinarDate)
       as DATE) as EndOfService
from Webinars as w
INNER JOIN (
    select
        WebinarID,
        WebinarDate,
```

```
        DurationTime
    from Webinars -- assuming WebinarDate and DurationTime are in the Webinars table
) as T1
    on T1.WebinarID = w.WebinarID
group by
    w.ServiceID,
    T1.WebinarDate,
    T1.DurationTime;
```

5.0.23 Widok STUDIES_INCOME - Emil Żychowicz

Widok przedstawia dane o przychodach generowanych przez studia:

- **StudiesID** – identyfikator studiów.
- **StudiesName** – nazwa studiów.
- **Income** – suma przychodów dla danej usługi studiów.

```
CREATE VIEW STUDIES_INCOME as
SELECT s.StudiesID, s.StudiesName, si.Income
FROM Studies AS s
INNER JOIN SERVICE_ID_INCOME AS si
    ON s.ServiceID = si.ServiceID
```

5.0.24 Widok STUDIES_USERS - Jakub Kaliński

Widok przedstawia informacje o użytkownikach zapisanych na studia:

- **StudiesID** – identyfikator studiów.
- **ServiceID** – identyfikator usługi związanej ze studiami.
- **StudiesName** – nazwa studiów.
- **ServiceUserID** – identyfikator użytkownika zapisującego się na studia.
- **FirstName** – imię użytkownika.
- **LastName** – nazwisko użytkownika.

```
create view STUDIES_USERS as
select sd.StudiesID, s.ServiceID, s.StudiesName, us.ServiceUserID, u.FirstName,
    u.LastName
  from StudiesDetails as sd
  INNER JOIN Studies as s
  on s.StudiesID = sd.StudiesID
  INNER JOIN ServiceUserDetails as us
  on us.ServiceUserID = sd.StudentID
  INNER JOIN Users as u
  on u.UserID = us.ServiceUserID
```

5.0.25 Widok WEBINAR_USERS - Jakub Kaliński

Widok przedstawia informacje o użytkownikach biorących udział w webinarach:

- **WebinarID** – identyfikator webinaru.
- **ServiceID** – identyfikator usługi związanej z webinarem.
- **WebinarName** – nazwa webinaru.
- **ServiceUserID** – identyfikator użytkownika uczestniczącego w webinarze.
- **FirstName** – imię użytkownika.
- **LastName** – nazwisko użytkownika.

```
create view WEBINAR_USERS as
select wd.WebinarID, w.ServiceID, w.WebinarName, us.ServiceUserID, u.FirstName,
    u.LastName
  from WebinarDetails as wd
```

```
  INNER JOIN Webinars as w
  on wd.WebinarID = w.WebinarID
  INNER JOIN ServiceUserDetails as us
  on us.ServiceUserID = wd.UserID
  INNER JOIN Users as u
  on u.UserID = us.ServiceUserID
```

5.0.26 Widok WEBINAR_INCOME - Emil Żychowicz

Widok przedstawia przychody związane z webinarami, połączone z odpowiednimi usługami.

- **WebinarID** – identyfikator webinaru.
- **WebinarName** – nazwa webinaru.
- **Income** – przychód związany z danym webinaru.

Widok łączy dane z tabeli **Webinars** i **SERVICE_ID_INCOME**, na podstawie identyfikatora usługi, aby wyświetlić przychód przypisany do każdego webinaru.

```
CREATE VIEW WEBINAR_INCOME AS
select w.WebinarID, w.WebinarName, si.Income
from Webinars as w
INNER JOIN SERVICE_ID_INCOME AS si
    ON w.ServiceID = si.ServiceID
```

5.0.27 Widok COURSE_SCHEDULE - Emil Żychowicz

Widok przedstawia harmonogram spotkań w kursach, z uwzględnieniem spotkań stacjonarnych oraz online. Spotkania są połączone z kursami i modułami.

- **CourseID** – identyfikator kursu.
- **CourseName** – nazwa kursu.
- **ModuleID** – identyfikator modułu.
- **MeetingID** – identyfikator spotkania.
- **MeetingType** – typ spotkania (np. "Stationary stacjonarne, "Online online).
- **StartOfMeeting** – data i godzina rozpoczęcia spotkania.
- **EndOfMeeting** – data i godzina zakończenia spotkania.

Widok łączy informacje z tabeli **COURSE_INFO** z danymi o spotkaniach stacjonarnych i online z tabeli **CombinedMeetings**.

```
CREATE OR ALTER VIEW COURSE_SCHEDULE AS
WITH CombinedMeetings AS (
    SELECT
        MeetingID,
        MeetingDate,
        MeetingDuration
    FROM StationaryMeeting
    UNION ALL
    SELECT
        MeetingID,
        MeetingDate,
        MeetingDuration
    FROM OnlineLiveMeeting
)
SELECT
  CI.CourseID,
  CI.CourseName,
    CI.ModuleID,
    CI.MeetingID,
    CI.MeetingType,
    CM.MeetingDate AS StartOfMeeting,
    DATEADD(MINUTE, DATEDIFF(MINUTE, '00:00:00', CM.MeetingDuration), CM.MeetingDate) AS
        EndOfMeeting
```

```
FROM COURSE_INFO CI
INNER JOIN CombinedMeetings CM
    ON CI.MeetingID = CM.MeetingID
WHERE CI.MeetingType != 'Offline Video';
```

5.0.28 Widok COURSE_PASSING_STATUS - Emil Żychowicz

Widok przedstawia status zaliczenia kursów dla uczestników na podstawie obecności na spotkaniach oraz ukończenia modułów. Obecność uczestników na kursach jest oceniana na podstawie spotkań stacjonarnych i online.

- **ServiceUserID** – identyfikator użytkownika serwisowego (uczestnika kursu).
- **CourseID** – identyfikator kursu.
- **ServiceID** – identyfikator usługi.
- **FirstName** – imię uczestnika.
- **LastName** – nazwisko uczestnika.
- **TotalModules** – całkowita liczba modułów w kursie.
- **PassedModules** – liczba modułów ukończonych pomyślnie.
- **CompletionPercentage** – procent ukończonych modułów.
- **CourseCompletion** – status ukończenia kursu ("PASSED"lub "NOT PASSED").

Widok oblicza status ukończenia kursu na podstawie liczby zaliczonych modułów oraz obecności na spotkaniach.

```
--stan zaliczenia osob zapisanych na zakonczone kursy
--czyli 100% obecnosci na spotkaniach z 80% modulow
create view COURSE_PASSING_STATUS as
SELECT
    T1.ServiceUserID,
    T1.CourseID,
    T1.ServiceID,
    T1.FirstName,
    T1.LastName,
    COUNT(T1.ModuleID) AS TotalModules,
    SUM(CASE WHEN T1.ModulePassed = 'True' THEN 1 ELSE 0 END) AS PassedModules,
    ROUND(CAST(SUM(CASE WHEN T1.ModulePassed = 'True' THEN 1 ELSE 0 END) AS FLOAT) /
        COUNT(T1.ModuleID) * 100,2) AS CompletionPercentage,
  CASE WHEN ROUND(CAST(SUM(CASE WHEN T1.ModulePassed = 'True' THEN 1 ELSE 0 END) AS
      FLOAT) / COUNT(T1.ModuleID) * 100,2) > 80 THEN 'PASSED' ELSE 'NOT PASSED' END as
      CourseCompletion
FROM (
    SELECT
        cu.ServiceUserID,
        cu.CourseID,
        alc.ModuleID,
        cu.ServiceID,
        cu.FirstName,
        cu.LastName,
        CASE
            WHEN
      (NOT EXISTS (select * from ATTENDANCE_LISTS_COURSES as a where a.ModuleID =
          alc.ModuleID) OR EXISTS (
                SELECT 1
                FROM ATTENDANCE_LISTS_COURSES as alc1
                WHERE alc1.ParticipantID = cu.ServiceUserID
                  AND alc1.CourseID = cu.CourseID
                  AND alc1.ModuleID = alc.ModuleID
                GROUP BY alc1.ParticipantID, alc1.CourseID, alc1.ModuleID
                HAVING COUNT(*) = SUM(CAST(alc1.Attendance AS INT)) -- czy byl na kazdym
                    spotkaniu nie offlinevideo
            ))
      AND
      (NOT EXISTS (select * from OfflineVideo as off_v where off_v.ModuleID =
          alc.ModuleID) OR EXISTS(  --czy byl na kazdym spotkaniu offline/ nie bylo
          spotkan offline
```

```
            SELECT 1
                  FROM ATTENDANCE_LISTS_OFFLINEVIDEO_COURSES as a_off
                  WHERE a_off.ParticipantID = cu.ServiceUserID
                    AND a_off.CourseID = cu.CourseID
                    AND a_off.ModuleID = alc.ModuleID
                  GROUP BY a_off.ParticipantID, a_off.CourseID, a_off.ModuleID
                  HAVING COUNT(*) = SUM(CAST(a_off.Attendance AS INT))
      ))
      THEN 'True'
            ELSE 'False'
        END AS ModulePassed
    FROM COURSES_USERS AS cu
    INNER JOIN
  (select CourseID, ModuleID from ATTENDANCE_LISTS_COURSES
  UNION
  select CourseID, ModuleID from ATTENDANCE_LISTS_OFFLINEVIDEO_COURSES
  ) as alc
        ON alc.CourseID = cu.CourseID
    GROUP BY
        cu.ServiceUserID,
        cu.CourseID,
        alc.ModuleID,
        cu.ServiceID,
        cu.FirstName,
        cu.LastName
) AS T1
INNER JOIN START_END_OF_COURSES as seoc
  on seoc.ServiceID = T1.ServiceID
WHERE seoc.StartOfService < CONVERT(DATE, GETDATE())
GROUP BY
    T1.ServiceUserID,
    T1.CourseID,
    T1.ServiceID,
    T1.FirstName,
    T1.LastName;



--SELECT
--    T1.ServiceUserID,
--    T1.CourseID,
--    T1.ServiceID,
--    T1.FirstName,
--    T1.LastName,
--    COUNT(T1.ModuleID) AS TotalModules,
--    SUM(CASE WHEN T1.ModulePassed = 'True' THEN 1 ELSE 0 END) AS PassedModules,
--    ROUND(CAST(SUM(CASE WHEN T1.ModulePassed = 'True' THEN 1 ELSE 0 END) AS FLOAT) /
    COUNT(T1.ModuleID) * 100,2) AS CompletionPercentage,
--  dbo.f_CheckIfCourseIsPassed(T1.ServiceUserID,T1.CourseID) as CourseCompletion,
--  CASE WHEN ROUND(CAST(SUM(CASE WHEN T1.ModulePassed = 'True' THEN 1 ELSE 0 END) AS
    FLOAT) / COUNT(T1.ModuleID) * 100,2) > 80 THEN 'PASSED' ELSE 'NOT PASSED' END as
    CourseCompletion
--FROM (
--    SELECT
--        cu.ServiceUserID,
--        cu.CourseID,
--        alc.ModuleID,
--        cu.ServiceID,
--        cu.FirstName,
--        cu.LastName,
--        CASE WHEN ROUND(dbo.f_CalculateAttendancePercentageOnModule(cu.ServiceUserID,
    cu.CourseID, alc.ModuleID), 0) = 100 THEN 'True' ELSE 'False' END AS ModulePassed
--    FROM COURSES_USERS AS cu
--    INNER JOIN
--  (select CourseID, ModuleID from ATTENDANCE_LISTS_COURSES
```

```
--  UNION
--  select CourseID, ModuleID from ATTENDANCE_LISTS_OFFLINEVIDEO_COURSES
--  ) as alc
--      ON alc.CourseID = cu.CourseID
--  GROUP BY
--          cu.ServiceUserID,
--          cu.CourseID,
--          alc.ModuleID,
--          cu.ServiceID,
--          cu.FirstName,
--          cu.LastName
--) AS T1
--INNER JOIN START_END_OF_COURSES as seoc
--  on seoc.ServiceID = T1.ServiceID
--WHERE seoc.StartOfService < CONVERT(DATE, GETDATE())
--GROUP BY
--    T1.ServiceUserID,
--    T1.CourseID,
--    T1.ServiceID,
--    T1.FirstName,
--    T1.LastName;
```

5.0.29 Widok COURSE_INFO - Emil Żychowicz

Widok zawiera informacje o kursach, w tym spotkania, moduły, daty rozpoczęcia i zakończenia kursu, oraz liczbę dostępnych miejsc.

- **CourseID** – identyfikator kursu.
- **ServiceID** – identyfikator usługi związanej z kursem.
- **CourseName** – nazwa kursu.
- **Vacancies** – liczba dostępnych miejsc w kursie (limit zapisów minus liczba zarejestrowanych użytkowników).
- **ModuleID** – identyfikator modułu.
- **ModuleType** – typ modułu (np. "Theory", "Practical").
- **StartOfService** – data rozpoczęcia kursu.
- **EndOfService** – data zakończenia kursu.
- **MeetingID** – identyfikator spotkania.
- **MeetingType** – typ spotkania (np. "Stationary", "Online Live", "Offline Video"). Widok łączy informacje z tabel **Courses**, **Modules**, **START_END_OF_COURSES**, i spotkań z różnych typów.

```
--informacje o kursie: spotkania z jakich sie sklada, moduly i ich typy, poczatek i
    koniec kursu, wolne miejsca na kurs.
create view COURSE_INFO as
select
  c.CourseID,
  c.ServiceID,
  c.CourseName,
  c.EnrollmentLimit - (select COUNT(*) from SERVICE_USERS where c.ServiceID =
      SERVICE_USERS.ServiceID) as Vacancies,
  m.ModuleID,
  m.ModuleType,
  c_start_end.StartOfService,
  c_start_end.EndOfService,
  T1.MeetingID,
  T1.MeetingType
from Courses as c
INNER JOIN Modules as m
  on m.CourseID = c.CourseID
INNER JOIN START_END_OF_COURSES as c_start_end
  on c_start_end.ServiceID = c.ServiceID
INNER JOIN (
  select meeting.MeetingID, meeting.ModuleID, 'Stationary' as MeetingType
  from StationaryMeeting as meeting
```

```
  union
  select meeting.MeetingID, meeting.ModuleID, 'Online Live' as MeetingType
  from OnlineLiveMeeting as meeting
  union
  select meeting.MeetingID, meeting.ModuleID, 'Offline Video' as MeetingType
  from OfflineVideo as meeting
) as T1
  on m.ModuleID = T1.ModuleID
```

5.0.30 Widok CONSUMER_BASKET - Emil Żychowicz

Widok przedstawia szczegóły koszyka konsumenta, uwzględniając zamówienia, usługi, ich wartości oraz status gotowości do udziału.

- **UserID** – identyfikator użytkownika.
- **OrderID** – identyfikator zamówienia.
- **ServiceID** – identyfikator usługi w zamówieniu.
- **ServiceType** – typ usługi.
- **ServiceValue** – wartość usługi.
- **AlreadyPaidForService** – kwota już opłacona za usługę.
- **IsReadyToParticipate** – informacja o gotowości do udziału w usłudze.

Widok łączy informacje z tabel **OrderDetails**, **Orders** oraz funkcji użytkowych w bazie danych, takich jak **f_GetServiceValue**, **f_CalculatePaidServiceValue**, i **f_IsReadyToParticipate**.

```
CREATE OR ALTER VIEW CONSUMER_BASKET AS
SELECT
  o.UserID,
  od.OrderID,
  od.ServiceID,
  (SELECT ServiceType FROM Services WHERE ServiceID =  od.ServiceID) as ServiceType,
  dbo.f_GetServiceValue(o.UserID, od.ServiceID) as ServiceValue,
  dbo.f_CalculatePaidServiceValue(od.ServiceID, od.OrderID) as AlreadyPaidForService,
  dbo.f_IsReadyToParticipate(o.UserID, od.OrderID, od.ServiceID) as IsReadyToParticipate
FROM OrderDetails as od
INNER JOIN Orders as o
ON od.OrderID = o.OrderID
```

5.0.31 Widok CLASS_MEETINGS_INCOME - Emil Żychowicz

Widok przedstawia przychody z klasy, łącząc spotkania klasowe z przypisanymi przychodami z tabeli **SERVICE_ID_INCOME**.

- **ClassMeetingID** – identyfikator spotkania klasowego.
- **Income** – przychód przypisany do spotkania. Widok łączy dane z tabel **ClassMeeting** i **SERVICE_ID_INCOME**.

```
CREATE VIEW CLASS_MEETINGS_INCOME as
SELECT cm.ClassMeetingID, si.Income
FROM ClassMeeting as cm
INNER JOIN SERVICE_ID_INCOME AS si
    ON cm.ServiceID = si.ServiceID
```

5.0.32 Widok ATTENDANCE_RAPORT - Jakub Kaliński

Widok zawiera raporty o obecności na spotkaniach związanych z kursami i studiami. W zależności od typu usługi, widok łączy dane z różnych źródeł.

- **ServiceType** – typ usługi ("Course"dla kursów, "Studies"dla studiów).
- **MeetingID** – identyfikator spotkania.
- **MeetingDate** – data spotkania.
- **MeetingType** – typ spotkania.

- **Attendance** – liczba osób obecnych na spotkaniu.

Widok łączy dane z tabel **ATTENDANCE_MEETINGS_IN_COURSES** i **V_ClassAttendanceAggregate**.

```
create view ATTENDANCE_RAPORT as
select
  'Course' as ServiceType,
  c.MeetingID,
  c.MeetingDate,
  c.MeetingType,
  c.Attendance
from ATTENDANCE_MEETINGS_IN_COURSES as c
union all
select
  'Studies' as ServiceType,
  s.MeetingID,
  s.MeetingDate,
  s.MeetingType,
  s.Attendance
from V_ClassAttendanceAggregate as s
```

5.0.33 Widok V_ConventionUsers - Michał Szymocha

Widok zawiera listę użytkowników zapisanych na wszystkie spotkania w ramach danego zjazdu.

- **ServiceID** - ID serwisu danego zjazdu
- **UserID** - ID uczestnika

```
CREATE OR ALTER VIEW V_ConventionUsers
AS
WITH
AllSyncMeetings AS
(
    SELECT
        c.ServiceID,
        COUNT(DISTINCT cm.ClassMeetingID) AS TotalSyncRequired
    FROM Convention c
    JOIN ClassMeeting cm
        ON cm.SubjectID = c.SubjectID
    LEFT JOIN StationaryClass st
        ON st.MeetingID = cm.ClassMeetingID
    LEFT JOIN OnlineLiveClass ol
        ON ol.MeetingID = cm.ClassMeetingID
    WHERE (st.MeetingID IS NOT NULL OR ol.MeetingID IS NOT NULL)
      AND (
           COALESCE(st.StartDate, ol.StartDate)
           BETWEEN c.StartDate
               AND DATEADD(DAY, c.Duration, c.StartDate)
          )
    GROUP BY c.ServiceID
),

AllAsyncMeetings AS
(
    SELECT
        c.ServiceID,
        COUNT(DISTINCT cm.ClassMeetingID) AS TotalAsyncRequired
    FROM Convention c
    JOIN ClassMeeting cm
        ON cm.SubjectID = c.SubjectID
    JOIN OfflineVideoClass ov
        ON ov.MeetingID = cm.ClassMeetingID
    WHERE ov.StartDate
          BETWEEN c.StartDate
```

```
                AND DATEADD(DAY, c.Duration, c.StartDate)
    GROUP BY c.ServiceID
),

UserSyncCounts AS
(
    SELECT
        c.ServiceID,
        scd.StudentID,
        COUNT(DISTINCT scd.MeetingID) AS UserSyncCount
    FROM SyncClassDetails scd
    JOIN ClassMeeting cm
        ON cm.ClassMeetingID = scd.MeetingID
    JOIN Convention c
        ON c.SubjectID = cm.SubjectID
    LEFT JOIN StationaryClass st
        ON st.MeetingID = cm.ClassMeetingID
    LEFT JOIN OnlineLiveClass ol
        ON ol.MeetingID = cm.ClassMeetingID
    WHERE (st.MeetingID IS NOT NULL OR ol.MeetingID IS NOT NULL)
      AND COALESCE(st.StartDate, ol.StartDate)
          BETWEEN c.StartDate
              AND DATEADD(DAY, c.Duration, c.StartDate)
    GROUP BY c.ServiceID, scd.StudentID
),
UserAsyncCounts AS
(
    SELECT
        c.ServiceID,
        acd.StudentID,
        COUNT(DISTINCT acd.MeetingID) AS UserAsyncCount
    FROM AsyncClassDetails acd
    JOIN ClassMeeting cm
        ON cm.ClassMeetingID = acd.MeetingID
    JOIN Convention c
        ON c.SubjectID = cm.SubjectID
    JOIN OfflineVideoClass ov
        ON ov.MeetingID = cm.ClassMeetingID
    WHERE ov.StartDate
          BETWEEN c.StartDate
              AND DATEADD(DAY, c.Duration, c.StartDate)
    GROUP BY c.ServiceID, acd.StudentID
)

SELECT
    s.ServiceID,
    s.StudentID AS UserID
FROM UserSyncCounts s
JOIN UserAsyncCounts a
    ON a.ServiceID = s.ServiceID
   AND a.StudentID    = s.StudentID
JOIN AllSyncMeetings reqS
    ON reqS.ServiceID = s.ServiceID
JOIN AllAsyncMeetings reqA
    ON reqA.ServiceID = a.ServiceID
WHERE s.UserSyncCount   = reqS.TotalSyncRequired
--    AND a.UserAsyncCount   = reqA.TotalAsyncRequired;
GO
```

5.0.34 Widok V_StudentGrades - Michał Szymocha

Widok przedstawia informacje o ocenach studentów w ramach ich studiów:

- **ServiceUserID** – identyfikator uczestnika usługi (studenta).

- **StudentName** – pełne imię i nazwisko studenta.
- **StudiesID** – identyfikator programu studiów.
- **StudiesName** – nazwa programu studiów.
- **StudiesGradeValue** – wartość oceny przypisana do studiów.
- **StudiesGradeLabel** – etykieta oceny przypisana do studiów.

```sql
CREATE VIEW V_StudentGrades AS
SELECT
    st.ServiceUserID,
    CONCAT(u.FirstName, ' ', u.LastName) AS StudentName,
    sd.StudiesID,
    s.StudiesName,
    g.GradeValue AS StudiesGradeValue,
    g.GradeName  AS StudiesGradeLabel
FROM StudiesDetails sd
    JOIN ServiceUserDetails st
    ON sd.StudentID = st.ServiceUserID
    JOIN Users u
    ON u.UserID = st.ServiceUserID
    JOIN Studies s
    ON s.StudiesID = sd.StudiesID
    JOIN Grades g
    ON g.GradeID = sd.StudiesGrade;
```

5.0.35 Widok V_SemesterSubjectsConventions - Michał Szymocha

Widok przedstawia informacje o konwencjach związanych z przedmiotami w semestrach:

- **SemesterID** – identyfikator semestru.
- **StudiesID** – identyfikator programu studiów.
- **SubjectID** – identyfikator przedmiotu.
- **SubjectName** – nazwa przedmiotu.
- **ConventionStart** – data rozpoczęcia konwencji.
- **ConventionDays** – liczba dni trwania konwencji.
- **ConventionID** – identyfikator konwencji.

```sql
CREATE VIEW V_SemesterSubjectsConventions AS
SELECT
    sem.SemesterID,
    sem.StudiesID,
    ss.SubjectID,
    sub.SubjectName,
    c.ConventionID,
    c.StartDate AS ConventionStart,
    c.Duration AS ConventionDays
FROM SemesterDetails sem
    JOIN Studies s
    ON s.StudiesID = sem.StudiesID
    JOIN SubjectStudiesAssignment ss
    ON ss.StudiesID = s.StudiesID
    JOIN Subject sub
    ON sub.SubjectID = ss.SubjectID
    LEFT JOIN Convention c
    ON c.SemesterID = sem.SemesterID
        AND c.SubjectID = ss.SubjectID;
```

5.0.36 Widok V_StudiesEnrollment - Michał Szymocha

Widok przedstawia informacje o zapisach na studia oraz koordynatorach programów:

- **StudiesID** – identyfikator programu studiów.

- **StudiesName** – nazwa programu studiów.
- **EnrollmentLimit** – limit zapisów na program studiów.
- **EnrollmentDeadline** – termin zapisów na program studiów.
- **TotalEnrolled** – całkowita liczba zapisanych studentów.
- **CoordinatorEmployeeID** – identyfikator pracownika koordynującego program.
- **CoordinatorName** – pełne imię i nazwisko koordynatora programu.

```sql
CREATE VIEW V_StudiesEnrollment AS
SELECT
    s.StudiesID,
    s.StudiesName,
    s.EnrollmentLimit,
    s.EnrollmentDeadline,
    COUNT(sd.StudentID) AS TotalEnrolled,
    e.EmployeeID AS CoordinatorEmployeeID,
    CONCAT(u.FirstName, ' ', u.LastName) AS CoordinatorName
FROM Studies s
    LEFT JOIN StudiesDetails sd
    ON s.StudiesID = sd.StudiesID
    LEFT JOIN Employees e
    ON e.EmployeeID = s.StudiesCoordinatorID
    LEFT JOIN Users u
    ON u.UserID = e.EmployeeID
GROUP BY
    s.StudiesID, s.StudiesName, s.EnrollmentLimit, s.EnrollmentDeadline,
    e.EmployeeID, u.FirstName, u.LastName;
```

5.0.37 Widok V_EmployeeHierarchy - Jakub Kaliński

Widok przedstawia strukturę hierarchiczną pracowników, wskazując przełożonych:

- **EmployeeID** – identyfikator pracownika.
- **EmployeeName** – pełne imię i nazwisko pracownika.
- **SuperiorID** – identyfikator przełożonego.
- **SuperiorName** – pełne imię i nazwisko przełożonego.

```sql
CREATE VIEW V_EmployeeHierarchy AS
SELECT
    e.EmployeeID,
    CONCAT(u.FirstName, ' ', u.LastName) AS EmployeeName,
    es.ReportsTo AS SuperiorID,
    CONCAT(u2.FirstName, ' ', u2.LastName) AS SuperiorName
FROM Employees e
    JOIN Users u
    ON u.UserID = e.EmployeeID
    LEFT JOIN EmployeesSuperior es
    ON es.EmployeeID = e.EmployeeID
    LEFT JOIN Employees e2
    ON e2.EmployeeID = es.ReportsTo
    LEFT JOIN Users u2
    ON u2.UserID = e2.EmployeeID;
```

5.0.38 Widok V_ConventionSchedule - Michał Szymocha

Widok przedstawia harmonogram konwencji związanych z przedmiotami:

- **ConventionID** – identyfikator konwencji.
- **SemesterID** – identyfikator semestru.
- **SubjectID** – identyfikator przedmiotu.
- **SubjectName** – nazwa przedmiotu.
- **StartDate** – data rozpoczęcia konwencji.

- **DurationDays** – liczba dni trwania konwencji.
- **EndDate** – data zakończenia konwencji (obliczona na podstawie `StartDate` i `DurationDays`).

```
CREATE VIEW V_ConventionSchedule AS
SELECT
    c.ConventionID,
    c.SemesterID,
    c.SubjectID,
    sub.SubjectName,
    c.StartDate,
    c.Duration AS DurationDays,
    DATEADD(DAY, c.Duration, c.StartDate) AS EndDate
FROM Convention c
    JOIN Subject sub
    ON c.SubjectID = sub.SubjectID;
```

5.0.39 Widok V_SubjectMeetingSchedule - Michał Szymocha

Widok przedstawia harmonogram spotkań związanych z przedmiotami:

- **ClassMeetingID** – identyfikator spotkania w ramach zajęć.
- **SubjectID** – identyfikator przedmiotu.
- **SubjectName** – nazwa przedmiotu.
- **MeetingType** – typ spotkania.
- **MeetingStartDate** – data rozpoczęcia spotkania.
- **TeacherID** – identyfikator nauczyciela prowadzącego spotkanie.
- **TeacherName** – pełne imię i nazwisko nauczyciela.
- **ConventionID** – identyfikator konwencji.
- **ConventionStart** – data rozpoczęcia konwencji.
- **ConventionDays** – liczba dni trwania konwencji.

```
CREATE VIEW V_SubjectMeetingSchedule AS
SELECT
    cm.ClassMeetingID,
    cm.SubjectID,
    s.SubjectName,
    cm.MeetingType,
    CASE WHEN sc.StartDate IS NOT NULL THEN sc.StartDate
         WHEN oc.StartDate IS NOT NULL THEN oc.StartDate
         ELSE ofc.StartDate
    END AS MeetingStartDate,
    t2.EmployeeID AS TeacherID,
    CONCAT(u2.FirstName, ' ', u2.LastName) AS TeacherName,
    c.ConventionID,
    c.StartDate AS ConventionStart,
    c.Duration AS ConventionDays
FROM ClassMeeting cm
    JOIN Subject s
    ON s.SubjectID = cm.SubjectID
    LEFT JOIN StationaryClass sc
    ON sc.MeetingID = cm.ClassMeetingID
    LEFT JOIN OnlineLiveClass oc
    ON oc.MeetingID = cm.ClassMeetingID
    LEFT JOIN OfflineVideoClass ofc
    ON ofc.MeetingID = cm.ClassMeetingID
    LEFT JOIN Employees t2
    ON t2.EmployeeID = cm.TeacherID
    LEFT JOIN Users u2
    ON u2.UserID = t2.EmployeeID
    LEFT JOIN Convention c
    ON c.SubjectID = cm.SubjectID
;
```

5.0.40 Widok V_TranslatorLanguageSkill - Jakub Kaliński

Widok przedstawia umiejętności językowe tłumaczy:

- **TranslatorID** – identyfikator tłumacza.
- **TranslatorName** – pełne imię i nazwisko tłumacza.
- **LanguageID** – identyfikator języka.
- **LanguageName** – nazwa języka.
- **Email** – adres email tłumacza.
- **Phone** – numer telefonu tłumacza.

```sql
CREATE VIEW V_TranslatorLanguageSkill AS
SELECT
    t.TranslatorID,
    CONCAT(u.FirstName, ' ', u.LastName) AS TranslatorName,
    l.LanguageID,
    l.LanguageName,
    uc.Email,
    uc.Phone
FROM Translators t
    JOIN Employees e
    ON e.EmployeeID = t.TranslatorID
    JOIN Users u
    ON u.UserID = e.EmployeeID
    JOIN TranslatorsLanguages tl
    ON tl.TranslatorID = t.TranslatorID
    JOIN Languages l
    ON l.LanguageID = tl.LanguageID
    LEFT JOIN UserContact uc
    ON uc.UserID = u.UserID;
```

5.0.41 Widok V_ConventionStudents - Michał Szymocha

Widok przedstawia informacje o studentach uczestniczących w konwencjach:

- **ConventionID** – identyfikator konwencji.
- **SemesterID** – identyfikator semestru.
- **SubjectID** – identyfikator przedmiotu.
- **SubjectName** – nazwa przedmiotu.
- **ServiceUserID** – identyfikator uczestnika usługi (studenta).
- **FirstName** – imię studenta.
- **LastName** – nazwisko studenta.
- **StartDate** – data rozpoczęcia konwencji.
- **ConventionDays** – liczba dni trwania konwencji.

```sql
CREATE VIEW V_ConventionStudents AS
SELECT
    c.ConventionID,
    c.SemesterID,
    c.SubjectID,
    sub.SubjectName,
    st.ServiceUserID,
    u.FirstName,
    u.LastName,
    c.StartDate,
    c.Duration AS ConventionDays
FROM Convention c
    JOIN SemesterDetails sem
    ON sem.SemesterID = c.SemesterID
    JOIN StudiesDetails sd
    ON sd.StudiesID = sem.StudiesID
    JOIN ServiceUserDetails st
    ON st.ServiceUserID = sd.StudentID
```

```
    JOIN Users u
    ON u.UserID = st.ServiceUserID
    JOIN SubjectStudiesAssignment s2s
    ON s2s.StudiesID = sem.StudiesID
        AND s2s.SubjectID = c.SubjectID
    JOIN Subject sub
    ON sub.SubjectID = c.SubjectID
;
```

5.0.42 Widok V_ClassMeetingStudents - Michał Szymocha

Widok przedstawia informacje o studentach przypisanych do spotkań w ramach zajęć:

- **ClassMeetingID** – identyfikator spotkania w ramach zajęć.
- **SubjectID** – identyfikator przedmiotu.
- **MeetingName** – nazwa spotkania.
- **MeetingType** – typ spotkania.
- **ServiceUserID** – identyfikator uczestnika usługi (studenta).
- **FirstName** – imię studenta.
- **LastName** – nazwisko studenta.
- **SyncAttendance** – informacja o obecności w przypadku zajęć synchronicznych.
- **AsyncViewDate** – data obejrzenia materiałów w przypadku zajęć asynchronicznych.

```
CREATE VIEW V_ClassMeetingStudents AS
SELECT
    cm.ClassMeetingID,
    cm.SubjectID,
    cm.MeetingName,
    cm.MeetingType,
    st.ServiceUserID,
    u.FirstName,
    u.LastName,
    scd.Attendance AS SyncAttendance,
    acd.ViewDate AS AsyncViewDate
FROM ClassMeeting cm
    LEFT JOIN SyncClassDetails scd
        ON scd.MeetingID = cm.ClassMeetingID
    LEFT JOIN AsyncClassDetails acd
        ON acd.MeetingID = cm.ClassMeetingID
    LEFT JOIN ServiceUserDetails st
        ON st.ServiceUserID = scd.StudentID
           OR st.ServiceUserID = acd.StudentID
    LEFT JOIN Users u
        ON u.UserID = st.ServiceUserID
WHERE scd.StudentID IS NOT NULL
      OR acd.StudentID IS NOT NULL
;
```

5.0.43 Widok AllEvents - Michał Szymocha

Widok przedstawia wszystkie wydarzenia (spotkania, webinarów) z informacjami o uczestnikach i czasie trwania:

- **ParticipantID** – identyfikator uczestnika.
- **EventType** – typ wydarzenia (np. StationaryClass, OnlineClass, Webinar).
- **EventID** – identyfikator wydarzenia.
- **StartTime** – czas rozpoczęcia wydarzenia.
- **EndTime** – czas zakończenia wydarzenia.

```
CREATE VIEW [dbo].[AllEvents] AS
    SELECT
```

```
        scd.StudentID as ParticipantID,
        'StationaryClass' AS EventType,
        sc.MeetingID        AS EventID,
        sc.MeetingDate      AS StartTime,
        DATEADD(Minute, 90, sc.MeetingDate) AS EndTime
    FROM StationaryMeeting sc
        JOIN SyncClassDetails scd
        ON sc.MeetingID = scd.MeetingID

UNION ALL

    SELECT
        scd.StudentID as ParticipantID,
        'OnlineClass' AS EventType,
        oc.MeetingID        AS EventID,
        oc.StartDate      AS StartTime,
        DATEADD(Minute, 90, oc.StartDate) AS EndTime
    FROM OnlineLiveClass oc
        JOIN SyncClassDetails scd
        ON oc.MeetingID = scd.MeetingID

UNION ALL

    SELECT
        smd.ParticipantID,
        'StationaryMeeting' AS EventType,
        sm.MeetingID        AS EventID,
        sm.MeetingDate      AS StartTime,
        DATEADD(Minute, 90, sm.MeetingDate) AS EndTime
    FROM StationaryMeeting sm
        JOIN StationaryMeetingDetails smd
        ON sm.MeetingID = smd.MeetingID

UNION ALL

    SELECT
        olmd.ParticipantID,
        'OnlineLiveMeeting' AS EventType,
        olm.MeetingID       AS EventID,
        olm.MeetingDate     AS StartTime,
        DATEADD(Minute, 90, olm.MeetingDate) AS EndTime
    FROM OnlineLiveMeeting olm
        JOIN OnlineLiveMeetingDetails olmd
        ON olm.MeetingID = olmd.MeetingID

UNION ALL
    SELECT
        wd.UserID   AS ParticipantID,
        'Webinar'    AS EventType,
        w.WebinarID  AS EventID,
        w.WebinarDate AS StartTime,
        DATEADD(Minute, 90, w.WebinarDate) AS EndTime
    FROM Webinars w
        JOIN WebinarDetails wd
        ON w.WebinarID = wd.WebinarID
;
```

5.0.44 Widok V_StudentCollidingEvents - Michał Szymocha

Widok przedstawia liczbę kolidujących wydarzeń dla każdego uczestnika:

- **ParticipantID** – identyfikator uczestnika.
- **CollisionsCount** – liczba kolizji (nakładających się) wydarzeń dla uczestnika.

```sql
CREATE VIEW V_StudentCollidingEvents AS
SELECT
    A.ParticipantID, A.EventID AS EID1, B.EventID as EID2, A.StartTime as ST1, A.EndTime
        as EnT1, B.StartTime as ST2, B.EndTime as EnT2
FROM AllEvents A
JOIN AllEvents B
    ON A.ParticipantID = B.ParticipantID
    AND A.EventID <> B.EventID
    AND A.StartTime < B.EndTime
    AND B.StartTime < A.EndTime
    AND A.EventID < B.EventID
GROUP BY
    A.ParticipantID, A.EventType, A.EventID, B.EventType, B.EventID, A.StartTime,
        A.EndTime, B.StartTime, B.EndTime
;
```

5.0.45 Widok V_StudentsFinishedStudies - Michał Szymocha

Widok przedstawia dane osób, które ukończyły studia:

- **ServiceUserID** – identyfikator użytkownika (studenta).
- **FirstName** – imię studenta.
- **LastName** – nazwisko studenta.
- **Address** – adres studenta.
- **PostalCode** – kod pocztowy studenta.
- **LocationID** – identyfikator lokalizacji.
- **StudiesID** – identyfikator ukończonych studiów.
- **StudiesName** – nazwa ukończonych studiów.

```sql
CREATE VIEW V_StudentsFinishedStudies
AS
SELECT
    st.ServiceUserID,
    u.FirstName,
    u.LastName,
    uad.Address,
    uad.PostalCode,
    uad.LocationID,
    s.StudiesID,
    s.StudiesName
FROM ServiceUserDetails st
JOIN
(
    SELECT
        sd.StudentID,
        sts.StudiesID,
        count(*) as TotalSubjects
    FROM SubjectDetails sd
    JOIN SubjectStudiesAssignment sts
        ON sd.SubjectID = sts.SubjectID
    GROUP BY sd.StudentID, sts.StudiesID
    HAVING
        COUNT(*) =
        (
            SELECT COUNT(*)
            FROM SubjectStudiesAssignment X
            WHERE X.StudiesID = sts.StudiesID
        )
        AND MIN(sd.SubjectGrade) >= 3
) AS T
   ON T.StudentID = st.ServiceUserID
JOIN Studies s
   ON s.StudiesID = T.StudiesID
```

```
JOIN Users u
   ON u.UserID = st.ServiceUserID
Join UserAddressDetails uad
    ON uad.UserID = u.UserID
;
```

5.0.46 Widok V_WebinarsWithAttendance - Michał Szymocha

Widok przedstawia informacje o webinarach wraz z obecnością uczestników:

- **WebinarID** – identyfikator webinaru.
- **WebinarName** – nazwa webinaru.
- **WebinarDate** – data webinaru.
- **DurationTime** – czas trwania webinaru.
- **ParticipantID** – identyfikator uczestnika.
- **Attended** – informacja o obecności uczestnika (1 – obecny).

```
SELECT
    w.WebinarID,
    w.WebinarName,
    w.WebinarDate,
    w.DurationTime,
    wd.UserID AS ParticipantID,
    1 AS Attended
FROM Webinars w
JOIN WebinarDetails wd
    ON w.WebinarID = wd.WebinarID;
```

5.0.47 Widok V_StudentSchedule - Jakub Kaliński

Widok przedstawia harmonogram wydarzeń dla studentów:

- **ParticipantID** – identyfikator uczestnika (studenta).
- **EventType** – typ wydarzenia (np. StationaryMeeting, OnlineLiveMeeting, StationaryClass, OnlineLiveClass, Webinar).
- **EventID** – identyfikator wydarzenia.
- **StartTime** – czas rozpoczęcia wydarzenia.
- **EndTime** – czas zakończenia wydarzenia.

```
CREATE VIEW dbo.V_StudentSchedule AS
SELECT
        u.UserID AS ParticipantID,
        'StationaryMeeting' AS EventType,
        sm.MeetingID AS EventID,
        sm.MeetingDate AS StartTime,
        DATEADD(SECOND, DATEDIFF(SECOND, 0, sm.MeetingDuration), sm.MeetingDate) AS
            EndTime
    FROM StationaryMeeting sm
    JOIN StationaryMeetingDetails smd ON sm.MeetingID = smd.MeetingID
    JOIN Users u ON u.UserID = smd.ParticipantID
                AND u.UserTypeID = 1

    UNION ALL

    SELECT
        u.UserID AS ParticipantID,
        'OnlineLiveMeeting' AS EventType,
        olm.MeetingID AS EventID,
        olm.MeetingDate AS StartTime,
        DATEADD(SECOND, DATEDIFF(SECOND, 0, olm.MeetingDuration), olm.MeetingDate) AS
            EndTime
```

```
    FROM OnlineLiveMeeting olm
    JOIN OnlineLiveMeetingDetails omd ON omd.MeetingID = olm.MeetingID
    JOIN Users u ON u.UserID = omd.ParticipantID
                AND u.UserTypeID = 1

    UNION ALL

    SELECT
        u.UserID AS ParticipantID,
        'StationaryClass' AS EventType,
        sc.MeetingID AS EventID,
        sc.StartDate AS StartTime,
        DATEADD(MINUTE, 90, sc.StartDate) AS EndTime
    FROM StationaryClass sc
    JOIN SyncClassDetails scd ON sc.MeetingID = scd.MeetingID
    JOIN Users u ON u.UserID = scd.StudentID
                AND u.UserTypeID = 1

    UNION ALL

    SELECT
        u.UserID AS ParticipantID,
        'OnlineLiveClass' AS EventType,
        olc.MeetingID AS EventID,
        olc.StartDate AS StartTime,
        DATEADD(SECOND, DATEDIFF(SECOND, 0, olc.Duration), olc.StartDate) AS EndTime
    FROM OnlineLiveClass olc
    JOIN SyncClassDetails olcd ON olc.MeetingID = olcd.MeetingID
    JOIN Users u ON u.UserID = olcd.StudentID
                AND u.UserTypeID = 1

    UNION ALL

    SELECT
        u.UserID AS ParticipantID,
        'Webinar' AS EventType,
        w.WebinarID AS EventID,
        w.WebinarDate AS StartTime,
        DATEADD(SECOND, DATEDIFF(SECOND, 0, w.DurationTime), w.WebinarDate) AS EndTime
    FROM Webinars w
    JOIN WebinarDetails wd ON w.WebinarID = wd.WebinarID
    JOIN Users u ON u.UserID = wd.UserID
                AND u.UserTypeID = 1
;
```

5.0.48 Widok V_EmployeeSchedule - Jakub Kaliński

Widok przedstawia harmonogram wydarzeń dla pracowników:

- **EmployeeID** – identyfikator pracownika.
- **EventType** – typ wydarzenia (np. StationaryMeeting, OnlineLiveMeeting, StationaryClass, OnlineLiveClass, Webinar).
- **EventID** – identyfikator wydarzenia.
- **StartTime** – czas rozpoczęcia wydarzenia.
- **EndTime** – czas zakończenia wydarzenia.

```
CREATE VIEW dbo.V_EmployeeSchedule
AS
    SELECT
        u.UserID AS EmployeeID,
        'StationaryMeeting' AS EventType,
        sm.MeetingID        AS EventID,
        sm.MeetingDate      AS StartTime,
```

```
        DATEADD(SECOND, DATEDIFF(SECOND, 0, sm.MeetingDuration), sm.MeetingDate) AS
            EndTime
    FROM StationaryMeeting sm
    JOIN Users u
        ON u.UserID = sm.TeacherID
       AND u.UserTypeID = 2

    UNION ALL

    SELECT
        u.UserID AS EmployeeID,
        'OnlineLiveMeeting' AS EventType,
        olm.MeetingID       AS EventID,
        olm.MeetingDate     AS StartTime,
        DATEADD(SECOND, DATEDIFF(SECOND, 0, olm.MeetingDuration), olm.MeetingDate) AS
            EndTime
    FROM OnlineLiveMeeting olm
    JOIN Users u
        ON u.UserID = olm.TeacherID
       AND u.UserTypeID = 2

    UNION ALL

        SELECT
       u.UserID AS EmployeeID,
       'StationaryClass' AS EventType,
       cm.ClassMeetingID AS EventID,
       sc.StartDate      AS StartTime,
       DATEADD(SECOND, DATEDIFF(SECOND, 0, sc.Duration), sc.StartDate) AS EndTime
    FROM ClassMeeting cm
    JOIN StationaryClass sc
       ON sc.MeetingID = cm.ClassMeetingID
    JOIN Users u
       ON u.UserID = cm.TeacherID
       AND u.UserTypeID = 2

    UNION ALL

    SELECT
       u.UserID AS EmployeeID,
       'OnlineLiveClass' AS EventType,
       cm.ClassMeetingID AS EventID,
       olc.StartDate     AS StartTime,
       DATEADD(SECOND, DATEDIFF(SECOND, 0, olc.Duration), olc.StartDate) AS EndTime
    FROM ClassMeeting cm
    JOIN OnlineLiveClass olc
       ON olc.MeetingID = cm.ClassMeetingID
    JOIN Users u
       ON u.UserID = cm.TeacherID
       AND u.UserTypeID = 2

    UNION ALL

    SELECT
        u.UserID AS EmployeeID,
        'Webinar'       AS EventType,
        w.WebinarID     AS EventID,
        w.WebinarDate   AS StartTime,
        DATEADD(SECOND, DATEDIFF(SECOND, 0, w.DurationTime), w.WebinarDate) AS EndTime
    FROM Webinars w
    JOIN Users u
        ON u.UserID = w.TeacherID
       AND u.UserTypeID = 2
;
```

5.0.49 Widok V_ClassAttendanceList - Jakub Kaliński

Widok przedstawia listę obecności studentów na zajęciach:

- **MeetingID** – identyfikator spotkania.
- **StudentID** – identyfikator studenta.
- **MeetingType** – typ spotkania (Stationary lub OnlineLive).
- **FirstName** – imię studenta.
- **LastName** – nazwisko studenta.
- **Attendance** – informacja o obecności.
- **StartDate** – data rozpoczęcia spotkania.

```
CREATE VIEW dbo.V_ClassAttendanceList
AS
    SELECT
        scc.MeetingID,
        scd.StudentID,
        'Stationary' AS MeetingType,
        st.FirstName,
        st.LastName,
        scd.Attendance,
        scc.StartDate
    FROM StationaryClass scc
    JOIN SyncClassDetails scd
        ON scc.MeetingID = scd.MeetingID
    JOIN Users st
        ON st.UserID = scd.StudentID

    UNION ALL

    SELECT
        olc.MeetingID,
        scd.StudentID,
        'OnlineLive' AS MeetingType,
        st.FirstName,
        st.LastName,
        scd.Attendance,
        olc.StartDate
    FROM OnlineLiveClass olc
    JOIN SyncClassDetails scd
        ON olc.MeetingID = scd.MeetingID
    JOIN Users st
        ON st.UserID = scd.StudentID
;
```

5.0.50 Widok V_ClassAttendanceAggregate - Michał Szymocha

Widok przedstawia zagregowane dane dotyczące obecności na zajęciach:

- **MeetingID** – identyfikator spotkania.
- **MeetingDate** – data spotkania.
- **MeetingType** – typ spotkania (StationaryClass lub OnlineLive).
- **Attendance** – średnia frekwencja na spotkaniu.

```
CREATE VIEW dbo.V_ClassAttendanceAggregate
AS
    SELECT
        scc.MeetingID,
        scc.StartDate AS MeetingDate,
        'StationaryClass'  AS MeetingType,
        AVG(CAST(scd.Attendance AS DECIMAL(5,2))) AS Attendance
    FROM StationaryClass scc
    JOIN SyncClassDetails scd
```

```
        ON scc.MeetingID = scd.MeetingID
    GROUP BY
        scc.MeetingID,
        scc.StartDate

    UNION ALL

    SELECT
        olc.MeetingID,
        olc.StartDate AS MeetingDate,
        'OnlineLive'  AS MeetingType,
        AVG(CAST(scd.Attendance AS DECIMAL(5,2))) AS Attendance
    FROM OnlineLiveClass olc
    JOIN SyncClassDetails scd
        ON olc.MeetingID = scd.MeetingID
    GROUP BY
        olc.MeetingID,
        olc.StartDate
;
```

5.0.51 Widok V_EmployeeWorkload - Michał Szymocha

Widok przedstawia obciążenie pracowników w minutach podzielone na jednostki 45-minutowe:

- **EmployeeID** – identyfikator pracownika.
- **worked_hours** – liczba przepracowanych jednostek (45 minut) przez pracownika.

```
CREATE VIEW V_EmployeeWorkload AS
SELECT
    EmployeeID,
    DATEDIFF(MINUTE, StartTime, EndTime)/45 AS worked_hours
FROM V_EmployeeSchedule
GROUP BY
    EmployeeID,
    DATEDIFF(MINUTE, StartTime, EndTime);
```

5.0.52 Widok V_StudiesInfo - Michał Szymocha

Widok przedstawia podstawowe informacje o oferowanych studiach:

```
CREATE OR ALTER VIEW V_StudiesInfo AS
SELECT
    s.StudiesID,
    s.StudiesName,
    s.StudiesDescription,
    s.StudiesCoordinatorID,
    s.EnrollmentDeadline,
    s.EnrollmentLimit,
    s.ExpectedGraduationDate,
    CONCAT(u.FirstName, ' ', u.LastName) AS CoordinatorName
FROM Studies s
    LEFT JOIN Employees e
    ON e.EmployeeID = s.StudiesCoordinatorID
    LEFT JOIN Users u
    ON u.UserID = e.EmployeeID
where s.EnrollmentDeadline > GETDATE();
```

5.0.53 Widok V_FutureStudentSchedule - Michał Szymocha

Widok przedstawia plan przyszłych zajęć dla studentów:

```
CREATE OR ALTER VIEW V_FutureStudentSchedule AS
SELECT * from V_StudentSchedule
WHERE StartTime > GETDATE()
GO
```

6 Procedury

6.0.1 Procedura p_AddRoom - Michał Szymocha

Procedura `p_AddRoom` umożliwia dodanie nowego pokoju do tabeli `Rooms`. Weryfikuje poprawność danych wejściowych:

- Sprawdza, czy `@Capacity` (pojemność pokoju) jest większe od 0.
- Sprawdza, czy `@Floor` (numer piętra) nie jest liczbą ujemną.

Jeśli dane wejściowe są poprawne, procedura dodaje rekord do tabeli. W przypadku wystąpienia błędów transakcja jest wycofywana, a użytkownik otrzymuje komunikat o błędzie.

```
CREATE OR ALTER PROCEDURE p_AddRoom
(
@Capacity INT,
@Address VARCHAR(100),
@Floor INT,
@AccessibleForDisabled BIT
)
AS
BEGIN
SET NOCOUNT ON;
BEGIN TRY
            BEGIN TRANSACTION;
            IF @Capacity <= 0
            BEGIN
                  RAISERROR('Room capacity must be greater than 0.', 16, 1);
                  ROLLBACK TRANSACTION;
                  RETURN;
            END;
            IF @Floor < 0
            BEGIN
                  RAISERROR('Floor number cannot be negative.', 16, 2);
                  ROLLBACK TRANSACTION;
                  RETURN;
            END;
            INSERT INTO Rooms
                  (Capacity, Address, Floor, AccessibleForDisabled)
            VALUES
                  (@Capacity, @Address, @Floor, @AccessibleForDisabled);
            COMMIT TRANSACTION;
      END TRY
      BEGIN CATCH
            IF @@TRANCOUNT > 0
                  ROLLBACK TRANSACTION;
            THROW;
      END CATCH;
END;
GO
```

6.0.2 Procedura p_ReserveRoom - Michał Szymocha

Procedura `p_ReserveRoom` umożliwia rezerwację pokoju (RoomID) na określony dzień (@Date) i przedział czasowy (@StartTime do @EndTime).

Walidacje:

- Sprawdza, czy podany @RoomID istnieje w tabeli Rooms.
- Weryfikuje, czy @Date jest datą przyszłą.
- Sprawdza, czy @StartTime jest wcześniejszy niż @EndTime.
- Sprawdza, czy pokój jest dostępny w określonym zakresie czasowym.

Jeśli wszystkie warunki są spełnione, procedura rezerwuje pokój w tabeli RoomScheduleOnDate. W przeciwnym razie transakcja jest wycofywana, a użytkownik otrzymuje odpowiedni komunikat o błędzie.

```
CREATE OR ALTER PROCEDURE p_ReserveRoom
(
      @RoomID       INT,
      @Date         DATE  ,
      @StartTime TIME(0),
      @EndTime    TIME(0)
)
AS
BEGIN
      SET NOCOUNT ON;
      BEGIN TRY
            BEGIN TRANSACTION;
            IF NOT EXISTS (SELECT 1 FROM Rooms WHERE RoomID = @RoomID)
            BEGIN
                  RAISERROR('Invalid RoomID: no matching Room found.', 16, 1);
                  ROLLBACK TRANSACTION;
                  RETURN;
            END;
            IF @Date < GETDATE()
            BEGIN
                  RAISERROR('Date must be in the future.', 16, 2);
                  ROLLBACK TRANSACTION;
                  RETURN;
            END;
            IF @StartTime >= @EndTime
            BEGIN
                  RAISERROR('StartTime must be before EndTime.', 16, 3);
                  ROLLBACK TRANSACTION;
                  RETURN;
            END;
            IF EXISTS
            (
                  SELECT 1
                  FROM RoomScheduleOnDate rs
                  WHERE rs.RoomID = @RoomID
                     AND rs.ScheduleOnDate = @Date
                     AND (
                           (rs.StartTime < @EndTime AND rs.EndTime > @StartTime) OR
                           (rs.StartTime >= @StartTime AND rs.EndTime <= @EndTime)
                     )
            )
            BEGIN
                  RAISERROR('Room is not available for the specified time range.', 16,
    4);
                  ROLLBACK TRANSACTION;
                  RETURN;
            END;
            DECLARE @NextSlotID INT;
            IF NOT EXISTS (SELECT 1 FROM RoomScheduleOnDate)
            BEGIN
                  SET @NextSlotID = 1;
            END
            ELSE
            BEGIN
                  SELECT @NextSlotID = MAX(SlotID) + 1 FROM RoomScheduleOnDate;
            END;
            DECLARE @availability bit;
            SET @availability = 0;
```

```
58              INSERT INTO RoomScheduleOnDate
59                    (SlotID, RoomID, ScheduleOnDate, StartTime, EndTime, SlotAvailability)
60              VALUES
61                    (@NextSlotID, @RoomID, @Date, @StartTime, @EndTime, @availability);
62              COMMIT TRANSACTION;
63        END TRY
64        BEGIN CATCH
65              IF @@TRANCOUNT > 0
66                    ROLLBACK TRANSACTION;
67              THROW;
68        END CATCH;
69  END;
```

6.0.3 Procedura p_EditReservation - Michał Szymocha

Procedura `p_EditReservation` umożliwia edytowanie istniejących rezerwacji lub ich usuwanie w tabeli `RoomScheduleOnDate`. Weryfikuje poprawność danych wejściowych i zapewnia, że nowe godziny rezerwacji nie kolidują z istniejącymi. Procedura:

- Sprawdza, czy podany `SlotID`, `RoomID`, i `Date` istnieją.
- Weryfikuje poprawność godzin (`StartTime`, `EndTime`).
- Zapewnia brak kolizji z innymi rezerwacjami.
- Obsługuje usuwanie rezerwacji, jeśli parametr `@Remove` wynosi `1`.

```
1   CREATE or Alter Procedure p_EditReservation
2   (
3         @SlotID INT,
4         @RoomID INT,
5         @Date DATE,
6         @StartTime TIME(0) = NULL,
7         @EndTime TIME(0) = NULL,
8         @Remove BIT = 0
9   )
10  AS
11  BEGIN
12        SET NOCOUNT ON;
13        BEGIN TRY
14              BEGIN TRANSACTION;
15              IF NOT EXISTS (SELECT 1 FROM RoomScheduleOnDate WHERE SlotID = @SlotID and
        RoomID = @RoomID and ScheduleOnDate = @Date)
16              BEGIN
17                    RAISERROR('Invalid SlotID: no matching Slot found.', 16, 1);
18                    ROLLBACK TRANSACTION;
19                    RETURN;
20              END;
21              IF @StartTime IS NOT NULL OR @EndTime IS NOT NULL
22              BEGIN
23                    IF @StartTime is NULL
24                    BEGIN
25                          SET @StartTime = (SELECT StartTime FROM RoomScheduleOnDate WHERE
        SlotID = @SlotID and RoomID = @RoomID and ScheduleOnDate = @Date);
26                    END;
27                    IF @EndTime is NULL
28                    BEGIN
29                          SET @EndTime = (SELECT EndTime FROM RoomScheduleOnDate WHERE
        SlotID = @SlotID and RoomID = @RoomID and ScheduleOnDate = @Date);
30                    END;
31                    IF @StartTime >= @EndTime
32                    BEGIN
33                          RAISERROR('StartTime must be before EndTime.', 16, 4);
34                          ROLLBACK TRANSACTION;
35                          RETURN;
36                    END;
37                    IF EXISTS
```

```
                    (
                          SELECT 1
                          FROM RoomScheduleOnDate rs
                          WHERE rs.RoomID = @RoomID
                             AND rs.ScheduleOnDate = @Date
                             AND rs.SlotID <> @SlotID
                             AND (
                                   (rs.StartTime < @EndTime AND rs.EndTime > @StartTime) OR
                                   (rs.StartTime >= @StartTime AND rs.EndTime <= @EndTime)
                             )
                    )
                    BEGIN
                          RAISERROR('Room is not available for the specified time range.',
    16, 5);
                          ROLLBACK TRANSACTION;
                          RETURN;
                    END;
              END;
              IF @Remove = 1
              BEGIN
                    DELETE FROM RoomScheduleOnDate
                    WHERE SlotID = @SlotID;
              END
              ELSE
              BEGIN
                    UPDATE RoomScheduleOnDate
                    SET RoomID = @RoomID,
                          ScheduleOnDate = @Date,
                          StartTime = @StartTime,
                          EndTime = @EndTime
                    WHERE SlotID = @SlotID;
              END;
              COMMIT TRANSACTION;
        END TRY
        BEGIN CATCH
              IF @@TRANCOUNT > 0
                    ROLLBACK TRANSACTION;
              THROW;
        END CATCH;
END;
```

6.0.4 Procedura p_CreateStudies - Michał Szymocha

Procedura `p_CreateStudies` umożliwia tworzenie nowych kierunków studiów oraz ich semestrów i powiązanych praktyk. Procedura:

- Weryfikuje poprawność danych wejściowych, takich jak:
  - Istnienie koordynatora (`@CoordinatorID`).
  - Minimalna liczba semestrów (minimum 2).
  - Limit zapisów oraz termin zapisów.
- Automatycznie wylicza identyfikatory oraz daty rozpoczęcia i zakończenia semestrów.
- Tworzy rekordy w tabelach `Studies`, `SemesterDetails`, i `Internship`.

```
CREATE OR ALTER PROCEDURE p_CreateStudies

(
      @StudiesName              VARCHAR(200),
      @StudiesDescription   VARCHAR  (1000),
      @CoordinatorID          INT,
      @EnrollmentLimit        INT,
      @EnrollmentDeadline   DATE  ,
      @SemesterCount          INT
```

```
)
AS
BEGIN
      SET NOCOUNT ON;
      BEGIN TRY
            BEGIN TRANSACTION;
            IF NOT EXISTS (SELECT 1 FROM Employees WHERE EmployeeID = @CoordinatorID)
            BEGIN
                  RAISERROR('Invalid CoordinatorID: no matching Employee found.', 16, 1);
                  ROLLBACK TRANSACTION;
                  RETURN;
            END;
            IF @SemesterCount < 2
            BEGIN
                  RAISERROR('SemesterCount must be at least 2.', 16, 2);
                  ROLLBACK TRANSACTION;
                  RETURN;
            END;
            IF @EnrollmentLimit <= 0
            BEGIN
                  RAISERROR('EnrollmentLimit must be greater than 0.', 16, 3);
                  ROLLBACK TRANSACTION;
                  RETURN;
            END;
            IF @EnrollmentDeadline < GETDATE()
            BEGIN
                  RAISERROR('EnrollmentDeadline must be in the future.', 16, 4);
                  ROLLBACK TRANSACTION;
                  RETURN;
            END;
            DECLARE @NextStudiesID INT;
            IF NOT EXISTS (SELECT 1 FROM Studies)
            BEGIN
                  SET @NextStudiesID = 1;
            END
            ELSE
            BEGIN
                  SELECT @NextStudiesID = MAX(StudiesID) + 1 FROM Studies;
            END;
            INSERT INTO Studies
                  (StudiesID, StudiesName, StudiesDescription, StudiesCoordinatorID,
                    EnrollmentLimit   , EnrollmentDeadline, SemesterCount,
    ExpectedGraduationDate)
            VALUES
                  (
                        @NextStudiesID,
                        @StudiesName,
                        @StudiesDescription,
                        @CoordinatorID,
                        @EnrollmentLimit,
                        @EnrollmentDeadline,
                        @SemesterCount,
                        NULL
                  );
            DECLARE @SemesterIndex INT = 1;
            DECLARE @StartDate       DATE = DATEADD(DAY, 1, @EnrollmentDeadline);
            DECLARE @EndDate          DATE;
            DECLARE @InternshipStart DATE = NULL;
            DECLARE @NextSemesterID INT;
            IF NOT EXISTS (SELECT 1 FROM SemesterDetails)
            BEGIN
                  SET @NextSemesterID = 1;
            END
            ELSE
            BEGIN
```

```
                SELECT @NextSemesterID = MAX(SemesterID) + 1 FROM SemesterDetails;
        END;
        WHILE @SemesterIndex <= @SemesterCount
        BEGIN
                SET @EndDate = DATEADD(DAY, 120, @StartDate);
                INSERT INTO SemesterDetails
                        (SemesterID, StudiesID, StartDate, EndDate)
                VALUES
                        (
                                @NextSemesterID,
                                @NextStudiesID,
                                @StartDate,
                                @EndDate
                        );
                IF @SemesterIndex = @SemesterCount - 1
                BEGIN
                        SET @InternshipStart = @StartDate;
                END;
                SET @SemesterIndex += 1;
                SET @NextSemesterID += 1;
                IF @SemesterIndex <= @SemesterCount
                BEGIN
                        SET @StartDate = DATEADD(DAY, 30 , @EndDate);
                END;
        END;
        UPDATE Studies
                SET ExpectedGraduationDate = @EndDate
        WHERE StudiesID = @NextStudiesID;
        DECLARE @NextInternshipID INT;
        IF NOT EXISTS (SELECT 1 FROM Internship)
        BEGIN
                SET @NextInternshipID = 1;
        END
        ELSE
        BEGIN
                SELECT @NextInternshipID = MAX(InternshipID) + 1 FROM Internship;
        END;
        INSERT INTO Internship
                (InternshipID, StudiesID, StartDate)
        VALUES
                (
                        @NextInternshipID,
                        @NextStudiesID,
                        @InternshipStart
                );
        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH;
END;
GO
```

6.0.5 Procedura p_AddSubject - Michał Szymocha

Procedura `p_AddSubject` umożliwia dodanie nowego przedmiotu do bazy danych. Weryfikuje, czy dane wejściowe są poprawne, takie jak nazwa przedmiotu, identyfikator koordynatora oraz liczba spotkań. Tworzy nowy rekord w tabeli `Subject`, przypisując unikalny `SubjectID`.

Walidacje:

- Sprawdza, czy `SubjectName` nie jest pusty.
- Weryfikuje, czy `SubjectCoordinatorID` istnieje w tabeli `Employees`.

- Sprawdza, czy liczba spotkań (`Meetings`) nie jest ujemna.
- Generuje unikalny identyfikator przedmiotu (`SubjectID`).

```
CREATE OR ALTER PROCEDURE p_AddSubject
(
    @SubjectName            VARCHAR(100),
    @SubjectDescription     VARCHAR(500),
    @SubjectCoordinatorID   INT,
    @Meetings               INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF @SubjectName = ''
        BEGIN
            RAISERROR('SubjectName cannot be empty.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF NOT EXISTS (SELECT 1 FROM Employees WHERE EmployeeID = @SubjectCoordinatorID)
        BEGIN
            RAISERROR('Invalid CoordinatorID: no matching employee found.', 16, 2);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF @Meetings < 0
        BEGIN
            RAISERROR('Meetings cannot be negative.', 16, 3);
            ROLLBACK TRANSACTION;
            RETURN;
        END;


        DECLARE @NextSubjectID INT;
        IF NOT EXISTS (SELECT 1 FROM Subject)
        BEGIN
            SET @NextSubjectID = 1;
        END
        ELSE
        BEGIN
            SELECT @NextSubjectID = MAX(SubjectID) + 1
            FROM Subject;
        END;


        INSERT INTO Subject
            (SubjectID, SubjectName, SubjectDescription,
             SubjectCoordinatorID, Meetings)
        VALUES
            (@NextSubjectID,
             @SubjectName,
             @SubjectDescription,
             @SubjectCoordinatorID,
             @Meetings);

        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
```

```
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
GO
```

6.0.6 Procedura p_AddConvention - Michał Szymocha

Procedura `p_AddConvention` służy do dodania nowej konwencji (np. semestralnej) dla przedmiotu. Weryfikuje, czy dany semestr i przedmiot istnieją w bazie danych oraz czy data konwencji mieści się w przedziale czasowym semestru. Tworzy nowy rekord w tabeli `Convention`, przypisując unikalny `ConventionID`.

Walidacje:

- Sprawdza, czy `SemesterID` istnieje w tabeli `SemesterDetails`.
- Weryfikuje, czy `SubjectID` jest poprawne.
- Sprawdza, czy data konwencji mieści się w przedziale czasowym semestru.
- Generuje unikalny identyfikator konwencji (`ConventionID`).

```
CREATE OR ALTER PROCEDURE p_AddConvention
(
    @SubjectID INT,
    @SemesterID     INT,
    @ConventionDate DATE,
    @Duration       int
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM SemesterDetails WHERE SemesterID = @SemesterID)
        BEGIN
            RAISERROR('Invalid SemesterID: no matching semester.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF NOT EXISTS (SELECT 1 FROM Subject WHERE SubjectID = @SubjectID)
        BEGIN
            RAISERROR('Invalid SubjectID: no matching subject.', 16, 2);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF @Duration < 0
        BEGIN
            RAISERROR('Duration cannot be negative.', 16, 3);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        DECLARE @SemStart DATE, @SemEnd DATE;
        SELECT @SemStart = StartDate,
               @SemEnd   = EndDate
        FROM SemesterDetails
        WHERE SemesterID = @SemesterID;

        IF @ConventionDate < @SemStart OR DATEADD(DAY, @Duration, @ConventionDate) >
            @SemEnd
        BEGIN
```

```
            RAISERROR(
                'Convention date must fall within the semester start/end dates.',
                16, 4
            );
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        DECLARE @NextConventionID INT;
        IF NOT EXISTS (SELECT 1 FROM Convention)
        BEGIN
            SET @NextConventionID = 1;
        END
        ELSE
        BEGIN
            SELECT @NextConventionID = MAX(ConventionID) + 1 FROM Convention;
        END;

        INSERT INTO Convention
            (ConventionID, StartDate, SemesterID, Duration)
        VALUES
            (@NextConventionID, @ConventionDate, @SemesterID, @Duration);

        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
GO
```

6.0.7 Procedura p_EnrollStudentInStudies - Michał Szymocha

Procedura `p_EnrollStudentInStudies` dodaje studenta do programu studiów. Weryfikuje, czy student i studia istnieją w bazie danych, oraz czy student nie jest już zapisany na te studia. Tworzy nowy rekord w tabeli `StudiesDetails`, przypisując studenta do odpowiednich semestrów.

Walidacje:

- Sprawdza, czy `StudentID` istnieje w tabeli `ServiceUserDetails`.
- Weryfikuje, czy `StudiesID` istnieje w tabeli `Studies`.
- Sprawdza, czy student nie jest już zapisany na te studia.
- Dla każdego semestru przypisanego do studiów, dodaje rekord do tabeli `StudiesDetails`.

```
CREATE OR ALTER PROCEDURE p_EnrollStudentInStudies
(
    @StudentID INT,
    @StudiesID   INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM ServiceUserDetails WHERE ServiceUserID = @StudentID)
        BEGIN
            RAISERROR('Invalid StudentID: not found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
```

```
        END;

        IF NOT EXISTS (SELECT 1 FROM Studies WHERE StudiesID = @StudiesID)
        BEGIN
            RAISERROR('Invalid StudiesID: not found.', 16, 2);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF EXISTS (
            SELECT 1
            FROM StudiesDetails
            WHERE StudentID = @StudentID
              AND StudiesID   = @StudiesID
        )
        BEGIN
            RAISERROR('Student is already enrolled in this Study.', 16, 3);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        DECLARE @Enrolled INT = (SELECT COUNT(*) FROM StudiesDetails WHERE StudiesID =
            @StudiesID);
        IF @Enrolled >= (SELECT EnrollmentLimit FROM Studies WHERE StudiesID =
            @StudiesID)
        BEGIN
            RAISERROR('Enrollment limit has been reached.', 16, 4);
            ROLLBACK TRANSACTION;
            RETURN;
        END;
        DECLARE @SemesterCnt INT, @SemesterID INT, @FirstSemesterID INT;
        SET @SemesterCnt = (Select SemesterCount from Studies where StudiesID =
            @StudiesID);
        SET @FirstSemesterID = (Select MIN(SemesterID) from SemesterDetails where
            StudiesID = @StudiesID);
        SET @SemesterID = 1

        PRINT @SemesterCnt
        PRINT @FirstSemesterID
        PRINT @SemesterID

        While @SemesterID <= @SemesterCnt
        BEGIN
            INSERT INTO StudiesDetails
                (StudentID, StudiesID, StudiesGrade, SemesterID)
            VALUES
                (@StudentID, @StudiesID, 0, @SemesterID + @FirstSemesterID - 1);

            SET @SemesterID += 1;
        END;

        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
GO
```

6.0.8 Procedura p_CreateStationaryClass - Michał Szymocha

Procedura p_CreateStationaryClass umożliwia utworzenie klasy stacjonarnej dla określonego przedmiotu.
Walidacje:

- Sprawdza, czy wskazana konwencja obejmuje podany przedmiot oraz zakres dat.
- Weryfikuje dostępność pokoju w określonym czasie. Sprawdza istnienie nauczyciela oraz jego dostępność.
- W przypadku podanego tłumacza, weryfikuje jego istnienie oraz dostępność.
- Waliduje, czy język jest prawidłowy, jeśli został określony.
- Upewnia się, że @GroupSize jest większy od 0 i nie przekracza pojemności pokoju.
- Sprawdza, czy @StartDate jest w przyszłości.

Jeśli wszystkie warunki są spełnione, procedura dodaje rekordy do tabel związanych z klasami stacjonarnymi oraz usługami.

```
CREATE OR ALTER PROCEDURE p_CreateStationaryClass
(
    @SubjectID      INT,
    @TeacherID      INT,
    @MeetingName    VARCHAR(50),
    @TranslatorID   INT = NULL,
    @LanguageID     INT = NULL,
    @RoomID         INT,
    @GroupSize      INT,
    @StartDate      DATETIME,
    @Duration       TIME(0),
    @PriceStudents money,
    @PriceOthers    money
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        DECLARE @EndDate DATETIME;
        SET @EndDate = DATEADD(SECOND, DATEDIFF(SECOND, 0, @Duration), @StartDate);

        IF NOT EXISTS
        (
            SELECT 1
            FROM Convention c
            WHERE c.SubjectID = @SubjectID
              AND @StartDate >= c.StartDate
              AND @EndDate <= DATEADD(DAY, c.Duration, c.StartDate)
        )
        BEGIN
            RAISERROR(
                'No matching Convention covers this SubjectID and time range.',
                16, 1
            );
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF EXISTS
        (
            SELECT 1
            FROM RoomScheduleOnDate rs
            WHERE rs.RoomID = @RoomID
              AND rs.ScheduleOnDate = CAST(@StartDate AS DATE)
              AND (
                  (rs.StartTime <= CONVERT(TIME(0), @StartDate) AND rs.EndTime >
                      CONVERT(TIME(0), @StartDate)) OR
```

```
                    (rs.StartTime >= CONVERT(TIME(0), @StartDate) AND rs.EndTime <=
                        CONVERT(TIME(0), @EndDate))
                )
        )
        BEGIN
            RAISERROR('Room is not available for the specified time range.', 16, 2);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF NOT EXISTS (SELECT 1 FROM Subject WHERE SubjectID = @SubjectID)
        BEGIN
            RAISERROR('Invalid SubjectID: no matching Subject found.', 16, 3);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF NOT EXISTS (SELECT 1 FROM Users WHERE UserID = @TeacherID and UserTypeID = 2)
        BEGIN
            RAISERROR('Invalid TeacherID: no matching Employee found.', 16, 4);
            ROLLBACK TRANSACTION;
            RETURN;
        END;
        IF EXISTS
        (
            SELECT 1
            FROM V_EmployeeSchedule es
            WHERE es.EmployeeID = @TeacherID
              AND (
                  (es.StartTime < @EndDate AND es.EndTime > @StartDate)
              )
        )
        BEGIN
            RAISERROR('Teacher is not available for the specified time range.', 16, 4);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF @TranslatorID IS NOT NULL
        BEGIN
            IF NOT EXISTS (SELECT 1 FROM Users WHERE UserID = @TranslatorID and
                UserTypeID = 3)
            BEGIN
                RAISERROR('Invalid TranslatorID: no matching Employee found.', 16, 5);
                ROLLBACK TRANSACTION;
                RETURN;
            END;
            IF EXISTS
            (
                SELECT 1
                FROM V_EmployeeSchedule es
                WHERE es.EmployeeID = @TranslatorID
                  AND (
                      (es.StartTime < @EndDate AND es.EndTime > @StartDate)
                  )
            )
            BEGIN
                RAISERROR('Translator is not available for the specified time range.',
                    16, 6);
                ROLLBACK TRANSACTION;
                RETURN;
            END;
        END;

        IF @LanguageID IS NOT NULL
```

```
        BEGIN
            IF NOT EXISTS (SELECT 1 FROM Languages WHERE LanguageID = @LanguageID)
            BEGIN
                RAISERROR('Invalid LanguageID: no matching Language found.', 16, 7);
                ROLLBACK TRANSACTION;
                RETURN;
            END;
        END;

        IF @Duration < CAST('00:00:00' AS TIME)
        BEGIN
            RAISERROR('Duration cannot be negative.', 16, 9);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF @StartDate < GETDATE()
        BEGIN
            RAISERROR('StartDate must be in the future.', 16, 10);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF @GroupSize <= 0
        BEGIN
            RAISERROR('GroupSize must be greater than 0.', 16, 11);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        DECLARE @NextMeetingID INT;
        IF NOT EXISTS (SELECT 1 FROM ClassMeeting)
        BEGIN
            SET @NextMeetingID = 1;
        END
        ELSE
        BEGIN
            SELECT @NextMeetingID = MAX(ClassMeetingID) + 1 FROM ClassMeeting;
        END;

        DECLARE @ServiceID INT;
        SELECT @ServiceID = MAX(ServiceID) + 1 FROM Services;

        INSERT INTO ClassMeeting
            (ClassMeetingID, SubjectID, TeacherID, MeetingName,
             TranslatorID, LanguageID, ServiceID, MeetingType)
        VALUES
            (
                @NextMeetingID,
                @SubjectID,
                @TeacherID,
                @MeetingName,
                @TranslatorID,
                @LanguageID,
                @ServiceID,
                'Stationary'
            );

        INSERT INTO StationaryClass
            (MeetingID, RoomID, GroupSize, StartDate, Duration)
        VALUES
            (
                @NextMeetingID,
                @RoomID,
                @GroupSize,
```

```
                @StartDate,
                @Duration
            );

        INSERT INTO Services
            (ServiceID, ServiceType)
        VALUES
            (@ServiceID, 'ClassMeetingService');

        INSERT INTO ClassMeetingService
            (ServiceID, PriceStudents, PriceOthers)
        VALUES
            (@ServiceID, @PriceStudents, @PriceOthers);

        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
GO
```

6.0.9 Procedura p_CreateOnlineLiveClass - Michał Szymocha

Procedura p_CreateOnlineLiveClass służy do tworzenia klasy online transmitowanej na żywo.
Walidacje:

- Sprawdza, czy wskazana konwencja obejmuje podany przedmiot oraz zakres dat.
- Weryfikuje istnienie nauczyciela oraz jego dostępność.
- Jeśli podano tłumacza, sprawdza jego istnienie oraz dostępność.
- Upewnia się, że podano poprawny link do spotkania.
- Weryfikuje, czy @Duration nie jest ujemny.
- Sprawdza, czy @StartDate jest datą przyszłą.

Jeśli walidacje są poprawne, procedura dodaje nowe rekordy do tabel związanych z klasami online oraz usługami.

```
CREATE OR ALTER PROCEDURE p_CreateOnlineLiveClass
(
    @SubjectID      INT,
    @TeacherID      INT,
    @MeetingName    VARCHAR(50),
    @TranslatorID   INT = NULL,
    @LanguageID     INT = NULL,
    @Link           VARCHAR(50),
    @StartDate      DATETIME,
    @Duration       DATETIME,
    @PriceStudents  money,
    @PriceOthers    money
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        DECLARE @EndDate DATETIME;
        SET @EndDate = DATEADD(SECOND, DATEDIFF(SECOND, 0, @Duration), @StartDate);

```

```
        IF NOT EXISTS
        (
            SELECT 1
            FROM Convention c
            WHERE c.SubjectID = @SubjectID
              AND @StartDate >= c.StartDate
              AND @EndDate <= DATEADD(MINUTE, DATEDIFF(MINUTE, 0, c.Duration),
                  c.StartDate)
        )
        BEGIN
            RAISERROR(
                'No matching Convention covers this SubjectID and time range.',
                16, 1
            );
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF NOT EXISTS (SELECT 1 FROM Subject WHERE SubjectID = @SubjectID)
        BEGIN
            RAISERROR('Invalid SubjectID: no matching Subject found.', 16, 2);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF NOT EXISTS (SELECT 1 FROM Users WHERE UserID = @TeacherID and UserTypeID = 2)
        BEGIN
            RAISERROR('Invalid TeacherID: no matching Employee found.', 16, 3);
            ROLLBACK TRANSACTION;
            RETURN;
        END;
        IF EXISTS
        (
            SELECT 1
            FROM V_EmployeeSchedule es
            WHERE es.EmployeeID = @TeacherID
              AND (
                  (es.StartTime < @EndDate AND es.EndTime > @StartDate)
              )
        )
        BEGIN
            RAISERROR('Teacher is not available for the specified time range.', 16, 4);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF @TranslatorID IS NOT NULL
        BEGIN
            IF NOT EXISTS (SELECT 1 FROM Users WHERE UserID = @TranslatorID and
                UserTypeID = 3)
            BEGIN
                RAISERROR('Invalid TranslatorID: no matching Employee found.', 16, 5);
                ROLLBACK TRANSACTION;
                RETURN;
            END;
            IF EXISTS
            (
                SELECT 1
                FROM V_EmployeeSchedule es
                WHERE es.EmployeeID = @TranslatorID
                  AND (
                      (es.StartTime < @EndDate AND es.EndTime > @StartDate)
                  )
            )
            BEGIN
```

```
87                  RAISERROR('Translator is not available for the specified time range.',
                        16, 6);
88                  ROLLBACK TRANSACTION;
89                  RETURN;
90              END;
91          END;
92
93          IF @LanguageID IS NOT NULL
94          BEGIN
95              IF NOT EXISTS (SELECT 1 FROM Languages WHERE LanguageID = @LanguageID)
96              BEGIN
97                  RAISERROR('Invalid LanguageID: no matching Language found.', 16, 7);
98                  ROLLBACK TRANSACTION;
99                  RETURN;
100             END;
101         END;
102
103         IF @Link = ''
104         BEGIN
105             RAISERROR('Link cannot be empty.', 16, 8);
106             ROLLBACK TRANSACTION;
107             RETURN;
108         END;
109
110         IF @Duration < 0
111         BEGIN
112             RAISERROR('Duration cannot be negative.', 16, 9);
113             ROLLBACK TRANSACTION;
114             RETURN;
115         END;
116
117         IF @StartDate < GETDATE()
118         BEGIN
119             RAISERROR('StartDate must be in the future.', 16, 10);
120             ROLLBACK TRANSACTION;
121             RETURN;
122         END;
123
124         DECLARE @NextMeetingID INT;
125         IF NOT EXISTS (SELECT 1 FROM ClassMeeting)
126         BEGIN
127             SET @NextMeetingID = 1;
128         END
129         ELSE
130         BEGIN
131             SELECT @NextMeetingID = MAX(ClassMeetingID) + 1 FROM ClassMeeting;
132         END;
133
134         DECLARE @ServiceID INT;
135         SELECT @ServiceID = MAX(ServiceID) + 1 FROM Services;
136
137         INSERT INTO ClassMeeting
138             (ClassMeetingID, SubjectID, TeacherID, MeetingName,
139              TranslatorID, LanguageID, ServiceID, MeetingType)
140         VALUES
141             (
142                 @NextMeetingID,
143                 @SubjectID,
144                 @TeacherID,
145                 @MeetingName,
146                 @TranslatorID,
147                 @LanguageID,
148                 @ServiceID,
149                 'OnlineLiveClass'
150             );
```

```
151
152         INSERT INTO OnlineLiveClass
153             (MeetingID, Link, StartDate, Duration)
154         VALUES
155             (
156                 @NextMeetingID,
157                 @Link,
158                 @StartDate,
159                 @Duration
160             );
161
162         INSERT INTO Services
163             (ServiceID, ServiceType)
164         VALUES
165         (@ServiceID, 'ClassMeetingService');
166
167         INSERT INTO ClassMeetingService
168             (ServiceID, PriceStudents, PriceOthers)
169         VALUES
170             (@ServiceID, @PriceStudents, @PriceOthers);
171
172         COMMIT TRANSACTION;
173     END TRY
174
175     BEGIN CATCH
176         IF @@TRANCOUNT > 0
177             ROLLBACK TRANSACTION;
178
179         THROW;
180     END CATCH;
181 END;
182 GO
```

6.0.10 Procedura p_CreateOfflineVideoClass - Michał Szymocha

Procedura p_CreateOfflineVideoClass umożliwia utworzenie klasy offline opartej na materiale wideo.
Walidacje:

- Sprawdza, czy wskazana konwencja obejmuje podany przedmiot oraz zakres dat.
- Weryfikuje istnienie nauczyciela.
- Jeśli podano tłumacza, sprawdza jego istnienie.
- Upewnia się, że podano poprawny link do wideo.
- Sprawdza, czy @StartDate jest w przyszłości.
- Weryfikuje, czy @Deadline jest późniejszy niż @StartDate.

Jeśli wszystkie warunki są spełnione, procedura tworzy rekordy w tabelach powiązanych z klasami offline.

```
1  CREATE OR ALTER PROCEDURE p_CreateOfflineVideoClass
2  (
3      @SubjectID    INT,
4      @TeacherID    INT,
5      @MeetingName  VARCHAR(50),
6      @TranslatorID  INT = NULL,
7      @LanguageID    INT = NULL,
8      @VideoLink    VARCHAR(50),
9      @StartDate    DATETIME,
10     @Deadline     DATETIME,
11     @PriceStudents money,
12     @PriceOthers   money
13 )
14 AS
15 BEGIN
16     SET NOCOUNT ON;
```

```
17
18      BEGIN TRY
19          BEGIN TRANSACTION;
20
21          IF NOT EXISTS
22          (
23              SELECT 1
24              FROM Convention c
25              WHERE c.SubjectID = @SubjectID
26                AND @StartDate >= c.StartDate
27                AND @Deadline <= DATEADD(MINUTE, DATEDIFF(MINUTE, 0, c.Duration),
                      c.StartDate)
28          )
29          BEGIN
30              RAISERROR(
31                  'No matching Convention covers this SubjectID and time range.',
32                  16, 1
33              );
34              ROLLBACK TRANSACTION;
35              RETURN;
36          END;
37
38          IF NOT EXISTS (SELECT 1 FROM Subject WHERE SubjectID = @SubjectID)
39          BEGIN
40              RAISERROR('Invalid SubjectID: no matching Subject found.', 16, 2);
41              ROLLBACK TRANSACTION;
42              RETURN;
43          END;
44
45          IF NOT EXISTS (SELECT 1 FROM Users WHERE UserID = @TeacherID and UserTypeID = 2)
46          BEGIN
47              RAISERROR('Invalid TeacherID: no matching Employee found.', 16, 3);
48              ROLLBACK TRANSACTION;
49              RETURN;
50          END;
51
52          IF @TranslatorID IS NOT NULL
53          BEGIN
54              IF NOT EXISTS (SELECT 1 FROM Users WHERE UserID = @TranslatorID and
                    UserTypeID = 3)
55              BEGIN
56                  RAISERROR('Invalid TranslatorID: no matching Employee found.', 16, 4);
57                  ROLLBACK TRANSACTION;
58                  RETURN;
59              END;
60          END;
61
62          IF @LanguageID IS NOT NULL
63          BEGIN
64              IF NOT EXISTS (SELECT 1 FROM Languages WHERE LanguageID = @LanguageID)
65              BEGIN
66                  RAISERROR('Invalid LanguageID: no matching Language found.', 16, 5);
67                  ROLLBACK TRANSACTION;
68                  RETURN;
69              END;
70          END;
71
72          IF @VideoLink = ''
73          BEGIN
74              RAISERROR('VideoLink cannot be empty.', 16, 6);
75              ROLLBACK TRANSACTION;
76              RETURN;
77          END;
78
79          IF @StartDate < GETDATE()
```

```
        BEGIN
            RAISERROR('StartDate must be in the future.', 16, 7);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF @Deadline < @StartDate
        BEGIN
            RAISERROR('Deadline must be after StartDate.', 16, 8);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        DECLARE @NextMeetingID INT;
        IF NOT EXISTS (SELECT 1 FROM ClassMeeting)
        BEGIN
            SET @NextMeetingID = 1;
        END
        ELSE
        BEGIN
            SELECT @NextMeetingID = MAX(ClassMeetingID) + 1 FROM ClassMeeting;
        END;

        DECLARE @ServiceID INT;
        SELECT @ServiceID = MAX(ServiceID) + 1 FROM Services;

        INSERT INTO ClassMeeting
            (ClassMeetingID, SubjectID, TeacherID, MeetingName,
             TranslatorID, LanguageID, ServiceID, MeetingType)
        VALUES
            (
                @NextMeetingID,
                @SubjectID,
                @TeacherID,
                @MeetingName,
                @TranslatorID,
                @LanguageID,
                @ServiceID,
                'OfflineVideo'
            );

        INSERT INTO OfflineVideoClass
            (MeetingID, VideoLink, StartDate, Deadline)
        VALUES
            (
                @NextMeetingID,
                @VideoLink,
                @StartDate,
                @Deadline
            );

        INSERT INTO Services
            (ServiceID, ServiceType)
        VALUES
            (@ServiceID, 'ClassMeetingService');

        INSERT INTO ClassMeetingService
            (ServiceID, PriceStudents, PriceOthers)
        VALUES
            (@ServiceID, @PriceStudents, @PriceOthers);

        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
```

```
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
GO
```

6.0.11 Procedura p_ChangeSubjectCoordinator - Michał Szymocha

Procedura p_ChangeSubjectCoordinator umożliwia zmianę koordynatora przedmiotu.
Walidacje:
Sprawdza, czy podany @NewCoordinatorID istnieje w tabeli Employees.
Weryfikuje, czy przedmiot o podanym @SubjectID istnieje w tabeli Subject.
Jeśli walidacje są poprawne, procedura aktualizuje identyfikator koordynatora w tabeli Subject. W przypadku błędów transakcja jest wycofywana, a użytkownik otrzymuje komunikat o błędzie.

```
CREATE OR ALTER PROCEDURE p_ChangeSubjectCoordinator
(
    @SubjectID           INT,
    @NewCoordinatorID    INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM Employees WHERE EmployeeID = @NewCoordinatorID)
        BEGIN
            RAISERROR('Invalid NewCoordinatorID: no matching Employee found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF NOT EXISTS (SELECT 1 FROM Subject WHERE SubjectID = @SubjectID)
        BEGIN
            RAISERROR('Invalid SubjectID: no matching Subject found.', 16, 2);
            ROLLBACK TRANSACTION;
            RETURN;
        END;
        Update Subject
        Set SubjectCoordinatorID = @NewCoordinatorID
        Where SubjectID = @SubjectID;

        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.12 Procedura p_ChangeStudiesCoordinator - Michał Szymocha

Ta procedura składowana zmienia koordynatora dla konkretnego programu studiów. Wykonuje następujące sprawdzenia:

- Weryfikuje, czy nowy koordynator istnieje w bazie danych.

- Upewnia się, że program studiów istnieje w bazie danych.
- Aktualizuje identyfikator koordynatora programu studiów na nowy identyfikator koordynatora.

```
CREATE OR ALTER PROCEDURE p_ChangeStudiesCoordinator
(
    @StudiesID           INT,
    @NewCoordinatorID    INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM Employees WHERE EmployeeID = @NewCoordinatorID)
        BEGIN
            RAISERROR('Invalid NewCoordinatorID: no matching Employee found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF NOT EXISTS (SELECT 1 FROM Studies WHERE StudiesID = @StudiesID)
        BEGIN
            RAISERROR('Invalid StudiesID: no matching Studies found.', 16, 2);
            ROLLBACK TRANSACTION;
            RETURN;
        END;
        Update Studies
        Set StudiesCoordinatorID = @NewCoordinatorID
        Where StudiesID = @StudiesID;

        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.13 Procedura p_EditStudies - Michał Szymocha

Ta procedura składowana pozwala na edytowanie szczegółów konkretnego programu studiów. Umożliwia modyfikację następujących atrybutów:

- Termin zapisów na studia.
- Limit liczby zapisanych studentów.
- Liczba semestrów.
- Opis studiów.
- Oczekiwana data ukończenia studiów.
- Nazwa programu studiów.

Procedura sprawdza, czy wartości są poprawne, np. czy termin zapisów nie jest przeszły, czy liczba semestrów jest większa niż 1, czy nazwa i opis nie są puste.

```
CREATE OR ALTER PROCEDURE p_EditStudies
(
    @StudiesID INT,
    @EnrollmentDeadline DATE = NULL,
    @EnrollmentLimit INT = NULL,
    @SemesterCnt INT = NULL,
```

```
    @StudiesDescription VARCHAR(50) = NULL,
    @ExpectedGraduationDate DATE = NULL,
    @StudiesName VARCHAR(50) = NULL
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM Studies WHERE StudiesID = @StudiesID)
        BEGIN
            RAISERROR('Invalid StudiesID: no matching Studies found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF @EnrollmentDeadline IS NOT NULL
        BEGIN
            IF @EnrollmentDeadline < GETDATE()
            BEGIN
                RAISERROR('EnrollmentDeadline must be in the future.', 16, 2);
                ROLLBACK TRANSACTION;
                RETURN;
            END;
            UPDATE Studies
            SET EnrollmentDeadline = @EnrollmentDeadline
            WHERE StudiesID = @StudiesID;
        END;

        IF @EnrollmentLimit IS NOT NULL
        BEGIN
            IF @EnrollmentLimit <= 0
            BEGIN
                RAISERROR('EnrollmentLimit must be greater than 0.', 16, 3);
                ROLLBACK TRANSACTION;
                RETURN;
            END;
            UPDATE Studies
            SET EnrollmentLimit = @EnrollmentLimit
            WHERE StudiesID = @StudiesID;
        END;

        IF @SemesterCnt IS NOT NULL
        BEGIN
            IF @SemesterCnt < 2
            BEGIN
                RAISERROR('SemesterCount must be at least 2.', 16, 4);
                ROLLBACK TRANSACTION;
                RETURN;
            END;
            UPDATE Studies
            SET SemesterCount = @SemesterCnt
            WHERE StudiesID = @StudiesID;
        END;

        IF @StudiesDescription IS NOT NULL
        BEGIN
            IF @StudiesDescription = ''
            BEGIN
                RAISERROR('StudiesDescription cannot be empty.', 16, 5);
                ROLLBACK TRANSACTION;
                RETURN;
            END;
```

```
            UPDATE Studies
            SET StudiesDescription = @StudiesDescription
            WHERE StudiesID = @StudiesID;
        END;

        IF @ExpectedGraduationDate IS NOT NULL
        BEGIN
            IF @ExpectedGraduationDate < DATEADD(day, (Select SemesterCount from Studies
                where StudiesID = @StudiesID),GETDATE())
            BEGIN
                RAISERROR('ExpectedGraduationDate must be in the future.', 16, 6);
                ROLLBACK TRANSACTION;
                RETURN;
            END;
            UPDATE Studies
            SET ExpectedGraduationDate = @ExpectedGraduationDate
            WHERE StudiesID = @StudiesID;
        END;

        IF @StudiesName IS NOT NULL
        BEGIN
            IF @StudiesName = ''
            BEGIN
                RAISERROR('StudiesName cannot be empty.', 16, 7);
                ROLLBACK TRANSACTION;
                RETURN;
            END;
            UPDATE Studies
            SET StudiesName = @StudiesName
            WHERE StudiesID = @StudiesID;
        END;

        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.14 Procedura p_EditSubject - Michał Szymocha

Ta procedura składowana pozwala na edytowanie szczegółów konkretnego przedmiotu. Umożliwia modyfikację następujących atrybutów:

- Nazwa przedmiotu.
- Opis przedmiotu.
- Liczba spotkań związanych z przedmiotem.

Procedura sprawdza, czy przedmiot istnieje w bazie danych oraz czy wprowadzone wartości są poprawne, np. nazwa przedmiotu nie może być pusta, a liczba spotkań musi być liczbą dodatnią.

```
CREATE OR ALTER PROCEDURE p_EditSubject
(
    @SubjectID INT,
    @SubjectName VARCHAR(50) = NULL,
    @SubjectDescription VARCHAR(50) = NULL,
    @Meetings INT = NULL
)
AS
BEGIN
    SET NOCOUNT ON;
```

```
    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM Subject WHERE SubjectID = @SubjectID)
        BEGIN
            RAISERROR('Invalid SubjectID: no matching Subject found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF @SubjectName IS NOT NULL
        BEGIN
            if @SubjectName = ''
            BEGIN
                RAISERROR('SubjectName cannot be empty.', 16, 2);
                ROLLBACK TRANSACTION;
                RETURN;
            END;
            UPDATE Subject
            SET SubjectName = @SubjectName
            WHERE SubjectID = @SubjectID;
        END;

        IF @SubjectDescription IS NOT NULL
        BEGIN
            if @SubjectDescription = ''
            BEGIN
                RAISERROR('SubjectDescription cannot be empty.', 16, 2);
                ROLLBACK TRANSACTION;
                RETURN;
            END;
            UPDATE Subject
            SET SubjectDescription = @SubjectDescription
            WHERE SubjectID = @SubjectID;
        END;

        IF @Meetings IS NOT NULL
        BEGIN
            IF @Meetings <= 0
            BEGIN
                RAISERROR('Meetings must be positive.', 16, 4);
                ROLLBACK TRANSACTION;
                RETURN;
            END;
            UPDATE Subject
            SET Meetings = @Meetings
            WHERE SubjectID = @SubjectID;
        END;

        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.15 Procedura p_ChangeSubjectCoordinator - Michał Szymocha

Ta procedura zmienia koordynatora przedmiotu. Wykonuje następujące kroki:

1. Sprawdza, czy nowy koordynator (na podstawie jego ID) istnieje w bazie danych.
2. Weryfikuje, czy przedmiot (na podstawie jego ID) istnieje w bazie danych.
3. Zmienia koordynatora przedmiotu na nowego, jeśli wszystkie warunki są spełnione.

```
CREATE OR ALTER PROCEDURE p_ChangeSubjectCoordinator
(
    @SubjectID           INT,
    @NewCoordinatorID    INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM Employees WHERE EmployeeID = @NewCoordinatorID)
        BEGIN
            RAISERROR('Invalid NewCoordinatorID: no matching Employee found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF NOT EXISTS (SELECT 1 FROM Subject WHERE SubjectID = @SubjectID)
        BEGIN
            RAISERROR('Invalid SubjectID: no matching Subject found.', 16, 2);
            ROLLBACK TRANSACTION;
            RETURN;
        END;
        Update Subject
        Set SubjectCoordinatorID = @NewCoordinatorID
        Where SubjectID = @SubjectID;

        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.16 Procedura p_AddConvention - Michał Szymocha

Ta procedura dodaje nową konwencję dla przedmiotu w określonym semestrze. Proces obejmuje:

1. Sprawdzenie, czy podany semestr istnieje w bazie danych.
2. Weryfikację istnienia przedmiotu.
3. Walidację, że czas trwania konwencji nie jest negatywny.
4. Upewnienie się, że data konwencji mieści się w ramach daty rozpoczęcia i zakończenia semestru.
5. Dodanie nowej konwencji do tabeli `Convention`.

```
CREATE OR ALTER PROCEDURE p_AddConvention
(
    @SubjectID INT,
    @SemesterID     INT,
    @ConventionDate DATE,
    @Duration       int
)
AS
BEGIN
```

```
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM SemesterDetails WHERE SemesterID = @SemesterID)
        BEGIN
            RAISERROR('Invalid SemesterID: no matching semester.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF NOT EXISTS (SELECT 1 FROM Subject WHERE SubjectID = @SubjectID)
        BEGIN
            RAISERROR('Invalid SubjectID: no matching subject.', 16, 2);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF @Duration < 0
        BEGIN
            RAISERROR('Duration cannot be negative.', 16, 3);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        DECLARE @SemStart DATE, @SemEnd DATE;
        SELECT @SemStart = StartDate,
               @SemEnd   = EndDate
        FROM SemesterDetails
        WHERE SemesterID = @SemesterID;

        IF @ConventionDate < @SemStart OR DATEADD(DAY, @Duration, @ConventionDate) >
            @SemEnd
        BEGIN
            RAISERROR(
                'Convention date must fall within the semester start/end dates.',
                16, 4
            );
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        DECLARE @NextConventionID INT;
        IF NOT EXISTS (SELECT 1 FROM Convention)
        BEGIN
            SET @NextConventionID = 1;
        END
        ELSE
        BEGIN
            SELECT @NextConventionID = MAX(ConventionID) + 1 FROM Convention;
        END;

        INSERT INTO Convention
            (ConventionID, StartDate, SemesterID, Duration)
        VALUES
            (@NextConventionID, @ConventionDate, @SemesterID, @Duration);

        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

```

```
        THROW;
    END CATCH;
END;
```

6.0.17 Procedura p_AddSubjectToStudies - Michał Szymocha

Procedura ta dodaje przedmiot do programu studiów. Jej działania obejmują:

1. Weryfikację istnienia przedmiotu i programu studiów.
2. Sprawdzenie, czy przedmiot już nie jest przypisany do tego programu studiów.
3. Dodanie przypisania przedmiotu do programu studiów, jeśli spełnione są wszystkie warunki.

```
CREATE or alter procedure p_AddSubjectToStudies
(
    @SubjectID INT,
    @StudiesID INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM Subject WHERE SubjectID = @SubjectID)
        BEGIN
            RAISERROR('Invalid SubjectID: no matching Subject found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF NOT EXISTS (SELECT 1 FROM Studies WHERE StudiesID = @StudiesID)
        BEGIN
            RAISERROR('Invalid StudiesID: no matching Studies found.', 16, 2);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF EXISTS (
            SELECT 1
            FROM SubjectStudiesAssignment
            WHERE SubjectID = @SubjectID
              AND StudiesID = @StudiesID
        )
        BEGIN
            RAISERROR('Subject is already added to this Studies.', 16, 3);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        INSERT INTO SubjectStudiesAssignment
            (SubjectID, StudiesID)
        VALUES
            (@SubjectID, @StudiesID);

        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
```

```
END;
```

6.0.18 Procedura p_ChangeClassMeetingTeacher - Michał Szymocha

Procedura zmienia nauczyciela dla spotkania klasowego. Wykonuje:

1. Sprawdzenie, czy nowy nauczyciel (na podstawie jego ID) istnieje w bazie danych i ma odpowiedni typ użytkownika (nauczyciel).
2. Walidację, czy dane spotkanie klasowe istnieje.
3. Zaktualizowanie spotkania klasowego, przypisując nowego nauczyciela.

```
CREATE OR ALTER PROCEDURE p_ChangeClassMeetingTeacher
(
    @MeetingID INT,
    @NewTeacherID INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM Users WHERE UserID = @NewTeacherID and UserTypeID =
            2)
        BEGIN
            RAISERROR('Invalid NewTeacherID: no matching Employee found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF NOT EXISTS (SELECT 1 FROM ClassMeeting WHERE ClassMeetingID = @MeetingID)
        BEGIN
            RAISERROR('Invalid MeetingID: no matching Meeting found.', 16, 2);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        UPDATE ClassMeeting
        SET TeacherID = @NewTeacherID
        WHERE ClassMeetingID = @MeetingID;

        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.19 Procedura p_ChangeClassMeetingTranslator - Michał Szymocha

Ta procedura zmienia tłumacza przypisanego do spotkania klasowego. Działania obejmują:

1. Sprawdzenie, czy nowy tłumacz (na podstawie jego ID) istnieje w bazie danych i ma odpowiedni typ użytkownika (tłumacz).
2. Walidację, czy spotkanie klasowe istnieje.
3. Sprawdzenie, czy tłumacz mówi w wymaganym języku (na podstawie powiązania języka z tłumaczem).

4. Zaktualizowanie spotkania klasowego, przypisując nowego tłumacza.

```
CREATE OR ALTER PROCEDURE p_ChangeClassMeetingTranslator
(
    @MeetingID INT,
    @NewTranslatorID INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM Users WHERE UserID = @NewTranslatorID and
            UserTypeID = 3)
        BEGIN
            RAISERROR('Invalid NewTranslatorID: no matching Employee found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF NOT EXISTS (SELECT 1 FROM ClassMeeting WHERE ClassMeetingID = @MeetingID)
        BEGIN
            RAISERROR('Invalid MeetingID: no matching Meeting found.', 16, 2);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        Declare @LanguageID INT;
        SET @LanguageID = (Select LanguageID from ClassMeeting where ClassMeetingID =
            @MeetingID);
        IF NOT EXISTS (SELECT 1 FROM TranslatorsLanguages WHERE LanguageID = @LanguageID
            and TranslatorID = @NewTranslatorID)
        BEGIN
            RAISERROR('Translator does not speak the required language.', 16, 3);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        UPDATE ClassMeeting
        SET TranslatorID = @NewTranslatorID
        WHERE ClassMeetingID = @MeetingID;

        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.20 Procedura p_ChangeClassMeetingLanguage - Michał Szymocha

Procedura zmienia język przypisany do spotkania klasowego. Proces obejmuje:

1. Sprawdzenie, czy nowy język (na podstawie jego ID) istnieje w bazie danych.
2. Weryfikację, czy dane spotkanie klasowe istnieje.
3. Walidację, czy tłumacz spotkania mówi w nowym języku.
4. Zaktualizowanie spotkania klasowego, przypisując nowy język.

```sql
CREATE OR ALTER PROCEDURE p_ChangeClassMeetingLanguage
(
    @MeetingID INT,
    @NewLanguageID INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM Languages WHERE LanguageID = @NewLanguageID)
        BEGIN
            RAISERROR('Invalid NewLanguageID: no matching Language found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF NOT EXISTS (SELECT 1 FROM ClassMeeting WHERE ClassMeetingID = @MeetingID)
        BEGIN
            RAISERROR('Invalid MeetingID: no matching Meeting found.', 16, 2);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        Declare @TranslatorID INT;
        SET @TranslatorID = (Select TranslatorID from ClassMeeting where ClassMeetingID
            = @MeetingID);
        IF NOT EXISTS (SELECT 1 FROM TranslatorsLanguages WHERE LanguageID =
            @NewLanguageID and TranslatorID = @TranslatorID)
        BEGIN
            RAISERROR('Translator does not speak the required language.', 16, 3);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        UPDATE ClassMeeting
        SET LanguageID = @NewLanguageID
        WHERE ClassMeetingID = @MeetingID;

        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.21 Procedura p_EditClassMeeting - Michał Szymocha

Procedura p_EditClassMeeting umożliwia edytowanie danych spotkania klasy, takich jak nazwa spotkania, typ spotkania oraz ceny dla studentów i innych. Weryfikuje poprawność danych wejściowych, w tym istnienie spotkania oraz poprawność wartości cenowych. Procedura:

- Sprawdza, czy podane `MeetingID` istnieje w tabeli `ClassMeeting`.
- Aktualizuje `MeetingName`, `MeetingType`, `PriceStudents` i `PriceOthers`, jeśli odpowiednie parametry są niepuste.
- Obsługuje transakcje i zwraca błąd w przypadku nieprawidłowych danych.

```sql
CREATE OR ALTER PROCEDURE p_EditClassMeeting
(
    @MeetingID INT,
    @MeetingName VARCHAR(50) = NULL,
    @MeetingType VARCHAR(50) = NULL,
    @PriceStudents MONEY = NULL,
    @PriceOthers MONEY = NULL
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM ClassMeeting WHERE ClassMeetingID = @MeetingID)
        BEGIN
            RAISERROR('Invalid MeetingID: no matching Meeting found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF @MeetingName IS NOT NULL
        BEGIN
            IF @MeetingName = ''
            BEGIN
                RAISERROR('MeetingName cannot be empty.', 16, 2);
                ROLLBACK TRANSACTION;
                RETURN;
            END;
            UPDATE ClassMeeting
            SET MeetingName = @MeetingName
            WHERE ClassMeetingID = @MeetingID;
        END;

        IF @MeetingType IS NOT NULL
        BEGIN
            IF @MeetingType = ''
            BEGIN
                RAISERROR('MeetingType cannot be empty.', 16, 4);
                ROLLBACK TRANSACTION;
                RETURN;
            END;
            UPDATE ClassMeeting
            SET MeetingType = @MeetingType
            WHERE ClassMeetingID = @MeetingID;
        END;

        IF @PriceStudents IS NOT NULL
        BEGIN
            IF @PriceStudents < 0
            BEGIN
                RAISERROR('PriceStudents cannot be negative.', 16, 5);
                ROLLBACK TRANSACTION;
                RETURN;
            END;
            UPDATE ClassMeetingService
            SET PriceStudents = @PriceStudents
            WHERE ServiceID = (SELECT ServiceID FROM ClassMeeting WHERE ClassMeetingID =
                @MeetingID);
        END;

        IF @PriceOthers IS NOT NULL
        BEGIN
            IF @PriceOthers < 0
```

```
            BEGIN
                RAISERROR('PriceOthers cannot be negative.', 16, 6);
                ROLLBACK TRANSACTION;
                RETURN;
            END;
            UPDATE ClassMeetingService
            SET PriceOthers = @PriceOthers
            WHERE ServiceID = (SELECT ServiceID FROM ClassMeeting WHERE ClassMeetingID =
                @MeetingID);
        END;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.22 Procedura p_EditStationaryClass - Michał Szymocha

Procedura p_EditStationaryClass umożliwia edytowanie danych spotkania w klasie stacjonarnej, takich jak pokój, wielkość grupy, data rozpoczęcia i czas trwania. Weryfikuje poprawność danych wejściowych oraz sprawdza dostępność pokoju. Procedura:

- Sprawdza, czy `MeetingID` istnieje w tabeli `ClassMeeting`.
- Weryfikuje poprawność `RoomID` i `GroupSize`, zapewniając, że rozmiar grupy nie przekracza pojemności pokoju.
- Aktualizuje dane dotyczące pokoju, grupy, daty rozpoczęcia i trwania spotkania.

```
CREATE OR ALTER PROCEDURE p_EditStationaryClass
(
    @MeetingID INT,
    @RoomID INT = NULL,
    @GroupSize INT = NULL,
    @StartDate DATETIME = NULL,
    @Duration DATETIME = NULL
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM ClassMeeting WHERE ClassMeetingID = @MeetingID)
        BEGIN
            RAISERROR('Invalid MeetingID: no matching Meeting found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF @RoomID IS NOT NULL
        BEGIN
            IF NOT EXISTS (SELECT 1 FROM Rooms WHERE RoomID = @RoomID)
            BEGIN
                RAISERROR('Invalid RoomID: no matching Room found.', 16, 2);
                ROLLBACK TRANSACTION;
                RETURN;
            END;
            UPDATE StationaryClass
            SET RoomID = @RoomID
```

```
            WHERE MeetingID = @MeetingID;
        END;

        IF @GroupSize IS NOT NULL
        BEGIN
            IF @GroupSize <= 0
            BEGIN
                RAISERROR('GroupSize must be greater than 0.', 16, 3);
                ROLLBACK TRANSACTION;
                RETURN;
            END;

            DECLARE @RoomCapacity INT;
            SET @RoomCapacity = (SELECT Capacity FROM Rooms WHERE RoomID = (SELECT
                RoomID FROM StationaryClass WHERE MeetingID = @MeetingID));

            IF @GroupSize > @RoomCapacity
            BEGIN
                RAISERROR('GroupSize cannot exceed Room capacity.', 16, 4);
                ROLLBACK TRANSACTION;
                RETURN;
            END;

            UPDATE StationaryClass
            SET GroupSize = @GroupSize
            WHERE MeetingID = @MeetingID;
        END;

        IF @StartDate IS NOT NULL
        BEGIN
            IF @StartDate < GETDATE()
            BEGIN
                RAISERROR('StartDate must be in the future.', 16, 5);
                ROLLBACK TRANSACTION;
                RETURN;
            END;
            UPDATE StationaryClass
            SET StartDate = @StartDate
            WHERE MeetingID = @MeetingID;
        END;

        IF @Duration IS NOT NULL
        BEGIN
            IF CAST(@Duration AS TIME(0)) < CAST('00:00:00' AS TIME(0))
            BEGIN
                RAISERROR('Duration cannot be negative.', 16, 6);
                ROLLBACK TRANSACTION;
                RETURN;
            END;

            UPDATE StationaryClass
            SET Duration = @Duration
            WHERE MeetingID = @MeetingID;
        END;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.23 Procedura p_EditOnlineLiveClass - Michał Szymocha

Procedura p_EditOnlineLiveClass umożliwia edytowanie danych spotkania online na żywo, takich jak link, data rozpoczęcia i czas trwania. Weryfikuje poprawność danych wejściowych oraz zapewnia, że data rozpoczęcia jest w przyszłości. Procedura:

- Sprawdza, czy `MeetingID` istnieje w tabeli `ClassMeeting`.
- Aktualizuje `Link`, `StartDate` i `Duration`, jeśli odpowiednie parametry są niepuste.
- Obsługuje transakcje i zapewnia, że daty są poprawne.

```sql
CREATE OR ALTER PROCEDURE p_EditOnlineLiveClass
(
    @MeetingID INT,
    @Link VARCHAR(50) = NULL,
    @StartDate DATETIME = NULL,
    @Duration DATETIME = NULL
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM ClassMeeting WHERE ClassMeetingID = @MeetingID)
        BEGIN
            RAISERROR('Invalid MeetingID: no matching Meeting found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF @Link IS NOT NULL
        BEGIN
            IF @Link = ''
            BEGIN
                RAISERROR('Link cannot be empty.', 16, 2);
                ROLLBACK TRANSACTION;
                RETURN;
            END;
            UPDATE OnlineLiveClass
            SET Link = @Link
            WHERE MeetingID = @MeetingID;
        END;

        IF @StartDate IS NOT NULL
        BEGIN
            IF @StartDate < GETDATE()
            BEGIN
                RAISERROR('StartDate must be in the future.', 16, 3);
                ROLLBACK TRANSACTION;
                RETURN;
            END;
            UPDATE OnlineLiveClass
            SET StartDate = @StartDate
            WHERE MeetingID = @MeetingID;
        END;

        IF @Duration IS NOT NULL
        BEGIN
            IF CAST(@Duration AS TIME(0)) < CAST('00:00:00' AS TIME(0))
            BEGIN
                RAISERROR('Duration cannot be negative.', 16, 4);
                ROLLBACK TRANSACTION;
                RETURN;
            END;
```

```
56
57              UPDATE OnlineLiveClass
58              SET Duration = @Duration
59              WHERE MeetingID = @MeetingID;
60          END;
61
62          COMMIT TRANSACTION;
63      END TRY
64      BEGIN CATCH
65          IF @@TRANCOUNT > 0
66              ROLLBACK TRANSACTION;
67
68          THROW;
69      END CATCH;
70  END;
```

6.0.24 Procedura Procedura p_EditOfflineVideoClass - Michał Szymocha

Procedura p_EditOfflineVideoClass umożliwia edytowanie danych spotkania offline wideo, takich jak link do wideo, data rozpoczęcia i termin ostateczny. Weryfikuje poprawność danych wejściowych, w tym relację między datą zakończenia a datą rozpoczęcia. Procedura:

- Sprawdza, czy `MeetingID` istnieje w tabeli `ClassMeeting`.
- Aktualizuje `VideoLink`, `StartDate` i `Deadline`, jeśli odpowiednie parametry są niepuste.
- Zapewnia, że termin ostateczny jest późniejszy niż data rozpoczęcia.

```
1   CREATE OR ALTER PROCEDURE p_EditOfflineVideoClass
2   (
3       @MeetingID INT,
4       @VideoLink VARCHAR(50) = NULL,
5       @StartDate DATETIME = NULL,
6       @Deadline DATETIME = NULL
7   )
8   AS
9   BEGIN
10      SET NOCOUNT ON;
11
12      BEGIN TRY
13          BEGIN TRANSACTION;
14
15          IF NOT EXISTS (SELECT 1 FROM ClassMeeting WHERE ClassMeetingID = @MeetingID)
16          BEGIN
17              RAISERROR('Invalid MeetingID: no matching Meeting found.', 16, 1);
18              ROLLBACK TRANSACTION;
19              RETURN;
20          END;
21
22          IF @VideoLink IS NOT NULL
23          BEGIN
24              IF @VideoLink = ''
25              BEGIN
26                  RAISERROR('VideoLink cannot be empty.', 16, 2);
27                  ROLLBACK TRANSACTION;
28                  RETURN;
29              END;
30              UPDATE OfflineVideoClass
31              SET VideoLink = @VideoLink
32              WHERE MeetingID = @MeetingID;
33          END;
34
35          IF @StartDate IS NOT NULL
36          BEGIN
37              IF @StartDate < GETDATE()
38              BEGIN
```

```
                RAISERROR('StartDate must be in the future.', 16, 3);
                ROLLBACK TRANSACTION;
                RETURN;
            END;
            UPDATE OfflineVideoClass
            SET StartDate = @StartDate
            WHERE MeetingID = @MeetingID;
        END;

        IF @Deadline IS NOT NULL
        BEGIN
            DECLARE @StartDt DATETIME;
            SELECT @StartDt = StartDate FROM OfflineVideoClass WHERE MeetingID =
                @MeetingID;
            IF @Deadline < @StartDt
            BEGIN
                RAISERROR('Deadline must be after StartDate.', 16, 4);
                ROLLBACK TRANSACTION;
                RETURN;
            END;

            UPDATE OfflineVideoClass
            SET Deadline = @Deadline
            WHERE MeetingID = @MeetingID;
        END;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.25 Procedura p_DeleteUserClassMeetingDetails - Michał Szymocha

Procedura p_DeleteUserClassMeetingDetails służy do usuwania szczegółów spotkania użytkownika z odpowiednich tabel, w zależności od typu spotkania. Procedura:

- Sprawdza, czy podane `MeetingID` istnieje w tabeli `ClassMeeting`.
- Zależnie od typu spotkania (`Stationary`, `OnlineLive`, `OfflineVideo`), usuwa odpowiednie szczegóły spotkania z tabel `SyncClassDetails`, `AsyncClassDetails`.
- Obsługuje transakcje i zapewnia, że operacja jest przeprowadzona prawidłowo.

```
CREATE or alter procedure p_DeleteUserClassMeetingDetails
(
    @MeetingID INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM ClassMeeting WHERE ClassMeetingID = @MeetingID)
        BEGIN
            RAISERROR('Invalid MeetingID: no matching Meeting found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

```

```
        DECLARE @MeetingType VARCHAR(50);
        SET @MeetingType = (SELECT MeetingType FROM ClassMeeting WHERE ClassMeetingID =
            @MeetingID);

        IF @MeetingType = 'Stationary' OR @MeetingType = 'OnlineLive'
        BEGIN
            DELETE FROM SyncClassDetails
            WHERE MeetingID = @MeetingID;
        END
        ELSE IF @MeetingType = 'OfflineVideo'
        BEGIN
            DELETE FROM AsyncClassDetails
            WHERE MeetingID = @MeetingID;
        END
        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.26 Procedura p_DeleteClassMeetingDetails - Michał Szymocha

Procedura p_DeleteClassMeetingDetails umożliwia usunięcie szczegółów spotkania z tabel związanych z określonym typem spotkania (stacjonarnym, online na żywo lub offline wideo). Procedura:

- Sprawdza, czy podane `MeetingID` istnieje w tabeli `ClassMeeting`.
- Usuwa szczegóły spotkania z odpowiednich tabel (np. `StationaryClass`, `OnlineLiveClass`, `OfflineVideoClass`).
- Wywołuje procedurę p_DeleteUserClassMeetingDetails w celu usunięcia użytkownika.
- Usuwa dane spotkania z tabeli `ClassMeeting`.

```
CREATE or alter procedure p_DeleteClassMeetingDetails
(
    @MeetingID INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM ClassMeeting WHERE ClassMeetingID = @MeetingID)
        BEGIN
            RAISERROR('Invalid MeetingID: no matching Meeting found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        DECLARE @MeetingType VARCHAR(50);
        SET @MeetingType = (SELECT MeetingType FROM ClassMeeting WHERE ClassMeetingID =
            @MeetingID);

        IF @MeetingType = 'Stationary'
        BEGIN
            DELETE FROM StationaryClass
            WHERE MeetingID = @MeetingID;
        END
        ELSE IF @MeetingType = 'OnlineLive'
        BEGIN
```

```
            DELETE FROM OnlineLiveClass
            WHERE MeetingID = @MeetingID;
        END
        ELSE IF @MeetingType = 'OfflineVideo'
        BEGIN
            DELETE FROM OfflineVideoClass
            WHERE MeetingID = @MeetingID;
        END

        exec [dbo].p_DeleteUserClassMeetingDetails @MeetingID;

        DELETE FROM ClassMeeting
        WHERE ClassMeetingID = @MeetingID;

        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.27 Procedura p_DeleteClassMeeting - Michał Szymocha

Procedura p_DeleteClassMeeting odpowiada za usunięcie całego spotkania z systemu, łącznie z usługami i szczegółami. Procedura:

- Sprawdza, czy `MeetingID` istnieje w tabeli `ClassMeeting`.
- Usuwa powiązane dane z tabel `ClassMeetingService` i `Services`.
- Usuwa szczegóły spotkania w zależności od jego typu (np. `StationaryClass`, `OnlineLiveClass`, `OfflineVideoClas`
- Wywołuje procedurę p_DeleteClassMeetingDetails w celu usunięcia szczegółów spotkania.
- Usuwa dane spotkania z tabeli `ClassMeeting`.

```
Create or alter PROCEDURE p_DeleteClassMeeting
(
    @MeetingID INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM ClassMeeting WHERE ClassMeetingID = @MeetingID)
        BEGIN
            RAISERROR('Invalid MeetingID: no matching Meeting found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        DECLARE @ServiceID INT;
        SET @ServiceID = (SELECT ServiceID FROM ClassMeeting WHERE ClassMeetingID =
            @MeetingID);

        DELETE FROM ClassMeetingService
        WHERE ServiceID = @ServiceID;

        DELETE FROM Services
        WHERE ServiceID = @ServiceID;

```

```
        DECLARE @MeetingType VARCHAR(50);
        SET @MeetingType = (SELECT MeetingType FROM ClassMeeting WHERE ClassMeetingID =
            @MeetingID);

        IF @MeetingType = 'Stationary'
        BEGIN
            DELETE FROM StationaryClass
            WHERE MeetingID = @MeetingID;
        END
        ELSE IF @MeetingType = 'OnlineLive'
        BEGIN
            DELETE FROM OnlineLiveClass
            WHERE MeetingID = @MeetingID;
        END
        ELSE IF @MeetingType = 'OfflineVideo'
        BEGIN
            DELETE FROM OfflineVideoClass
            WHERE MeetingID = @MeetingID;
        END

        exec [dbo].p_DeleteClassMeetingDetails @MeetingID;

        DELETE FROM ClassMeeting
        WHERE ClassMeetingID = @MeetingID;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.28 Procedura p_DeleteConvention - Michał Szymocha

Procedura p_DeleteConvention umożliwia usunięcie konwentu z systemu, łącznie z powiązanymi spotkaniami. Procedura:

- Sprawdza, czy `ConventionID` istnieje w tabeli `Convention`.
- Usuwa powiązane spotkania, z różnymi typami (np. `OfflineVideo`, `OnlineLive`, `Stationary`) w zależności od daty konwentu.
- Wywołuje procedurę p_DeleteClassMeeting w celu usunięcia spotkań.
- Usuwa dane konwentu z tabeli `Convention`.

```
CREATE OR ALTER PROCEDURE p_DeleteConvention
(
    @ConventionID INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM Convention WHERE ConventionID = @ConventionID)
        BEGIN
            RAISERROR('Invalid ConventionID: no matching Convention found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

```

```
        DECLARE @TargetSubject INT;
        SET @TargetSubject = (SELECT SubjectID FROM Convention WHERE ConventionID =
            @ConventionID);
        DECLARE @ConventionStartDate DATE, @ConventionEndDate DATE;
        SET @ConventionStartDate = (SELECT StartDate FROM Convention WHERE ConventionID
            = @ConventionID);
        SET @ConventionEndDate = DATEADD(DAY, (SELECT Duration FROM Convention WHERE
            ConventionID = @ConventionID), @ConventionStartDate);

        DECLARE @MeetingID INT;
        DECLARE OfflineCursor CURSOR FOR
        SELECT ClassMeetingID
        FROM ClassMeeting
        WHERE SubjectID = @TargetSubject
        AND MeetingType = 'OfflineVideo'
        AND ClassMeetingID IN (
            SELECT MeetingID
            FROM OfflineVideoClass
            WHERE Deadline >= @ConventionStartDate AND Deadline <= @ConventionEndDate
        );

        OPEN OfflineCursor;
        FETCH NEXT FROM OfflineCursor INTO @MeetingID;
        WHILE @@FETCH_STATUS = 0
        BEGIN
            EXEC [dbo].p_DeleteClassMeeting @MeetingID;
            FETCH NEXT FROM OfflineCursor INTO @MeetingID;
        END;
        CLOSE OfflineCursor;
        DEALLOCATE OfflineCursor;

        DECLARE OnlineCursor CURSOR FOR
        SELECT ClassMeetingID
        FROM ClassMeeting
        WHERE SubjectID = @TargetSubject
        AND MeetingType = 'OnlineLive'
        AND ClassMeetingID IN (
            SELECT MeetingID
            FROM OnlineLiveClass
            WHERE StartDate >= @ConventionStartDate AND StartDate <= @ConventionEndDate
        );

        OPEN OnlineCursor;
        FETCH NEXT FROM OnlineCursor INTO @MeetingID;
        WHILE @@FETCH_STATUS = 0
        BEGIN
            EXEC [dbo].p_DeleteClassMeeting @MeetingID;
            FETCH NEXT FROM OnlineCursor INTO @MeetingID;
        END;
        CLOSE OnlineCursor;
        DEALLOCATE OnlineCursor;

        DECLARE StationaryCursor CURSOR FOR
        SELECT ClassMeetingID
        FROM ClassMeeting
        WHERE SubjectID = @TargetSubject
        AND MeetingType = 'Stationary'
        AND ClassMeetingID IN (
            SELECT MeetingID
            FROM StationaryClass
            WHERE StartDate >= @ConventionStartDate AND StartDate <= @ConventionEndDate
        );

        OPEN StationaryCursor;
        FETCH NEXT FROM StationaryCursor INTO @MeetingID;
```

```
        WHILE @@FETCH_STATUS = 0
        BEGIN
            EXEC [dbo].p_DeleteClassMeeting @MeetingID;
            FETCH NEXT FROM StationaryCursor INTO @MeetingID;
        END;
        CLOSE StationaryCursor;
        DEALLOCATE StationaryCursor;
        DELETE FROM Convention
        WHERE ConventionID = @ConventionID;

        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.29 Procedura p_DeleteSubjectFromStudies - Michał Szymocha

Procedura p_DeleteSubjectFromStudies umożliwia usunięcie przypisania przedmiotu do studiów. Procedura:

- Sprawdza, czy podane `SubjectID` i `StudiesID` istnieją.
- Weryfikuje, czy przedmiot jest przypisany do studiów.
- Usuwa przypisanie przedmiotu do studiów z tabeli `SubjectStudiesAssignment`.

```
Create or alter procedure p_DeleteSubjectFromStudies
(
    @SubjectID INT,
    @StudiesID INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM Subject WHERE SubjectID = @SubjectID)
        BEGIN
            RAISERROR('Invalid SubjectID: no matching Subject found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF NOT EXISTS (SELECT 1 FROM Studies WHERE StudiesID = @StudiesID)
        BEGIN
            RAISERROR('Invalid StudiesID: no matching Studies found.', 16, 2);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF NOT EXISTS (
            SELECT 1
            FROM SubjectStudiesAssignment
            WHERE SubjectID = @SubjectID
              AND StudiesID = @StudiesID
        )
        BEGIN
            RAISERROR('Subject is not added to this Studies.', 16, 3);
```

```
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        DELETE FROM SubjectStudiesAssignment
        WHERE SubjectID = @SubjectID
          AND StudiesID = @StudiesID;

        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.30 Procedura p_DeleteSubject - Michał Szymocha

Procedura p_DeleteSubject umożliwia usunięcie przedmiotu z systemu. Procedura:

- Sprawdza, czy `SubjectID` istnieje w tabeli `Subject`.
- Usuwa dane przedmiotu z tabeli `Subject`.
- Usuwa powiązane dane konwentu, wywołując procedurę p_DeleteConvention.
- Usuwa przypisania przedmiotu do studiów z tabeli `SubjectStudiesAssignment`.

```
CREATE OR ALTER PROCEDURE p_DeleteSubject
(
    @SubjectID INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM Subject WHERE SubjectID = @SubjectID)
        BEGIN
            RAISERROR('Invalid SubjectID: no matching Subject found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        DELETE FROM Subject
        WHERE SubjectID = @SubjectID;

        DECLARE @ConventionID INT;
        DECLARE ConventionCursor CURSOR FOR
        SELECT ConventionID
        FROM Convention
        WHERE SubjectID = @SubjectID;

        OPEN ConventionCursor;
        FETCH NEXT FROM ConventionCursor INTO @ConventionID;
        WHILE @@FETCH_STATUS = 0
        BEGIN
            EXEC [dbo].p_DeleteConvention @ConventionID;
            FETCH NEXT FROM ConventionCursor INTO @ConventionID;
        END;
        CLOSE ConventionCursor;
        DEALLOCATE ConventionCursor;
```

```
37
38          DELETE FROM SubjectStudiesAssignment
39          WHERE SubjectID = @SubjectID;
40
41          COMMIT TRANSACTION;
42      END TRY
43
44      BEGIN CATCH
45          IF @@TRANCOUNT > 0
46              ROLLBACK TRANSACTION;
47
48          THROW;
49      END CATCH;
50  END;
```

6.0.31 Procedura p_CreateInternship - Michał Szymocha

Opisuje procedurę tworzenia nowego stażu, uwzględniając walidację studiów, daty rozpoczęcia oraz tworzenie unikalnego identyfikatora stażu.

```
1   CREATE OR ALTER PROCEDURE p_CreateInternship
2   (
3       @InternshipID INT,
4       @StudiesID INT,
5       @StartDate DATE
6   )
7   AS
8   BEGIN
9       SET NOCOUNT ON;
10
11      BEGIN TRY
12          BEGIN TRANSACTION;
13
14          IF NOT EXISTS (SELECT 1 FROM Studies WHERE StudiesID = @StudiesID)
15          BEGIN
16              RAISERROR('Invalid StudiesID: no matching Studies found.', 16, 1);
17              ROLLBACK TRANSACTION;
18              RETURN;
19          END;
20
21          IF @StartDate < GETDATE()
22          BEGIN
23              RAISERROR('StartDate must be in the future.', 16, 2);
24              ROLLBACK TRANSACTION;
25              RETURN;
26          END;
27
28          DECLARE @NextInternshipID INT;
29          IF NOT EXISTS (SELECT 1 FROM Internship)
30          BEGIN
31              SET @NextInternshipID = 1;
32          END
33          ELSE
34          BEGIN
35              SELECT @NextInternshipID = MAX(InternshipID) + 1 FROM Internship;
36          END;
37
38          INSERT INTO Internship
39              (InternshipID, StudiesID, StartDate)
40          VALUES
41              (@NextInternshipID, @StudiesID, @StartDate);
42
43          COMMIT TRANSACTION;
44      END TRY
45
```

```
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.32 Procedura p_EditInternship - Michał Szymocha

Procedura p_EditInternship umożliwia edytowanie szczegółów istniejącego stażu.
Walidacje:

- Sprawdza, czy @InternshipID istnieje w tabeli Internship.
- Jeśli podano @StartDate, weryfikuje, czy jest to data przyszła.

Jeśli warunki są spełnione, procedura aktualizuje datę rozpoczęcia stażu w tabeli Internship. W przeciwnym razie transakcja jest wycofywana, a użytkownik otrzymuje odpowiedni komunikat o błędzie.

```
CREATE OR ALTER PROCEDURE p_EditInternship
(
    @InternshipID INT,
    @StartDate DATE = NULL
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM Internship WHERE InternshipID = @InternshipID)
        BEGIN
            RAISERROR('Invalid InternshipID: no matching Internship found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF @StartDate IS NOT NULL
        BEGIN
            IF @StartDate < GETDATE()
            BEGIN
                RAISERROR('StartDate must be in the future.', 16, 2);
                ROLLBACK TRANSACTION;
                RETURN;
            END;
            UPDATE Internship
            SET StartDate = @StartDate
            WHERE InternshipID = @InternshipID;
        END;

        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.33 Procedura p_AddStudentInternship - Michał Szymocha

Procedura p_AddStudentInternship umożliwia przypisanie studenta do określonego stażu.
Walidacje:

- Sprawdza, czy @InternshipID istnieje w tabeli Internship.
- Weryfikuje, czy @StudentID istnieje w tabeli Users jako student.
- Sprawdza, czy student jest przypisany do programu studiów związanego z danym stażem.
- Weryfikuje, czy czas trwania stażu (@Duration) jest nieujemny.
- Upewnia się, że ocena (@InternshipGrade) jest prawidłowa.

Jeśli wszystkie warunki są spełnione, procedura dodaje rekord do tabeli InternshipDetails. W przeciwnym razie transakcja jest wycofywana.

```
Create or alter procedure p_AddStudentInternship
(
    @InternshipID INT,
    @StudiesID INT,
    @StudentID INT,
    @Duration INT = 14,
    @InternshipGrade INT = 0,
    @InternshipAttendance bit = NULL
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM Internship WHERE InternshipID = @InternshipID)
        BEGIN
            RAISERROR('Invalid InternshipID: no matching Internship found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF NOT EXISTS (SELECT 1 FROM Users WHERE UserID = @StudentID and UserTypeID = 1)
        BEGIN
            RAISERROR('Invalid StudentID: no matching Student found.', 16, 2);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF NOT EXISTS (SELECT 1 FROM StudiesDetails WHERE StudiesID = @StudiesID and
           StudentID = @StudentID)
        BEGIN
            RAISERROR('Invalid StudiesID: no matching Studies found for this student.',
                16, 3);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF EXISTS (
            SELECT 1
            FROM InternshipDetails
            WHERE InternshipID = @InternshipID
              AND StudentID = @StudentID
        )

        BEGIN
            RAISERROR('Student is already added to this Internship.', 16, 4);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

```

```
        IF @Duration < 0
        BEGIN
            RAISERROR('Duration cannot be negative.', 16, 5);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF @InternshipGrade NOT in (-1, 0, 1)
        BEGIN
            RAISERROR('Invalid InternshipGrade: no matching Grade found.', 16, 6);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        INSERT INTO InternshipDetails
            (InternshipID, StudentID, Duration, InternshipGrade, InternshipAttendance)
        VALUES
            (@InternshipID, @StudentID, @Duration, @InternshipGrade,
                @InternshipAttendance);

        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.34 Procedura p_InitiateInternship - Michał Szymocha

Procedura p_InitiateInternship inicjuje staż dla wszystkich studentów przypisanych do danego programu studiów.

Walidacje:

- Sprawdza, czy @InternshipID istnieje w tabeli Internship.
- Weryfikuje, czy @StudiesID istnieje w tabeli Studies.
- Upewnia się, że czas trwania stażu (@Duration) jest nieujemny.
- Weryfikuje, czy ocena (@InternshipGrade) jest prawidłowa.

Procedura przypisuje wszystkich studentów z danego programu studiów do stażu, tworząc wpisy w tabeli InternshipDetails. W przypadku błędów transakcja jest wycofywana.

```
Create or alter procedure p_InitiateInternship
(
    @InternshipID INT,
    @StudiesID INT,
    @Duration INT = 14,
    @InternshipGrade INT = 0,
    @InternshipAttendance bit = NULL
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM Internship WHERE InternshipID = @InternshipID)
        BEGIN
            RAISERROR('Invalid InternshipID: no matching Internship found.', 16, 1);
            ROLLBACK TRANSACTION;
```

```
            RETURN;
        END;

        IF NOT EXISTS (SELECT 1 FROM Studies WHERE StudiesID = @StudiesID)
        BEGIN
            RAISERROR('Invalid StudiesID: no matching Studies found.', 16, 2);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        DECLARE @StudentID INT;
        DECLARE StudentCursor CURSOR FOR
        SELECT StudentID
        FROM StudiesDetails
        WHERE StudiesID = @StudiesID;

        OPEN StudentCursor;
        FETCH NEXT FROM StudentCursor INTO @StudentID;
        WHILE @@FETCH_STATUS = 0
        BEGIN
            IF NOT EXISTS (
                SELECT 1
                FROM InternshipDetails
                WHERE InternshipID = @InternshipID
                  AND StudentID = @StudentID
            )
            BEGIN
                INSERT INTO InternshipDetails
                    (InternshipID, StudentID, Duration, InternshipGrade,
                        InternshipAttendance)
                VALUES
                    (@InternshipID, @StudentID, @Duration, @InternshipGrade,
                        @InternshipAttendance);
            END;
            FETCH NEXT FROM StudentCursor INTO @StudentID;
        END;
        CLOSE StudentCursor;
        DEALLOCATE StudentCursor;

        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.35 Procedura p_DeleteInternship - Michał Szymocha

Procedura p_DeleteInternship usuwa określony staż z bazy danych.
Walidacje:

- Sprawdza, czy @InternshipID istnieje w tabeli Internship.

Procedura usuwa rekord z tabeli Internship oraz wszystkie powiązane wpisy w tabeli InternshipDetails. Jeśli staż nie istnieje, transakcja jest wycofywana.

```
Create or Alter Procedure p_DeleteInternship
(
    @InternshipID INT
)
AS
```

```
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM Internship WHERE InternshipID = @InternshipID)
        BEGIN
            RAISERROR('Invalid InternshipID: no matching Internship found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        DELETE FROM InternshipDetails
        WHERE InternshipID = @InternshipID;

        DELETE FROM Internship
        WHERE InternshipID = @InternshipID;

        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.36 Procedura p_DeleteInternshipDetails - Michał Szymocha

Procedura p_DeleteInternshipDetails usuwa szczegóły stażu dla konkretnego studenta.
Walidacje:

- Sprawdza, czy @InternshipID istnieje w tabeli Internship.
- Weryfikuje, czy @StudentID istnieje w tabeli Users jako student.

Jeśli walidacje są spełnione, procedura usuwa odpowiedni rekord z tabeli InternshipDetails. W przeciwnym razie transakcja jest wycofywana.

```
CREATE OR ALTER PROCEDURE p_DeleteInternshipDetails
(
    @InternshipID INT,
    @StudentID INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM Internship WHERE InternshipID = @InternshipID)
        BEGIN
            RAISERROR('Invalid InternshipID: no matching Internship found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF NOT EXISTS (SELECT 1 FROM Users WHERE UserID = @StudentID and UserTypeID = 1)
        BEGIN
            RAISERROR('Invalid StudentID: no matching Student found.', 16, 2);
            ROLLBACK TRANSACTION;
            RETURN;
```

```
        END;

        IF NOT EXISTS (
            SELECT 1
            FROM InternshipDetails
            WHERE InternshipID = @InternshipID
              AND StudentID = @StudentID
        )
        BEGIN
            RAISERROR('Student is not added to this Internship.', 16, 3);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        DELETE FROM InternshipDetails
        WHERE InternshipID = @InternshipID
          AND StudentID = @StudentID;

        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.37 Procedura p_CreateGrade - Michał Szymocha

Procedura p_CreateGrade umożliwia dodanie nowej oceny dla studenta.
Walidacje:

- Sprawdza, czy @StudentID istnieje w tabeli Users jako student.
- Weryfikuje, czy wartość @GradeValue mieści się w dozwolonym zakresie.

Jeśli walidacje są spełnione, procedura dodaje rekord do tabeli Grades. W przeciwnym razie transakcja jest wycofywana.

```
CREATE OR ALTER PROCEDURE p_CreateGrade
(
    @GradeID INT,
    @GradeName VARCHAR(50),
    @GradeValue INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF @GradeValue < 0
        BEGIN
            RAISERROR('GradeValue cannot be negative.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF EXISTS (SELECT 1 FROM Grades WHERE GradeID = @GradeID)
        BEGIN
            RAISERROR('GradeID already exists.', 16, 2);
            ROLLBACK TRANSACTION;
            RETURN;
```

```
        END;

        INSERT INTO Grades
            (GradeID, GradeName, GradeValue)
        VALUES
            (@GradeID, @GradeName, @GradeValue);

        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.38 Procedura p_EnrollStudentInConvention - Michał Szymocha

Procedura p_EnrollStudentInConvention zapisuje studenta na określoną konwencję.
Walidacje:

- Sprawdza, czy @ConventionID istnieje w tabeli Convention.
- Weryfikuje, czy @StudentID istnieje w tabeli Users jako student.
- Sprawdza, czy student nie jest już zapisany na tę konwencję.

Jeśli walidacje są spełnione, procedura dodaje wpis do tabeli ConventionEnrollments. W przeciwnym razie transakcja jest wycofywana.

```
CREATE OR ALTER PROCEDURE p_EnrollStudentInConvention
(
    @UserID INT,
    @ConventionID INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM Convention WHERE ConventionID = @ConventionID)
        BEGIN
            RAISERROR('Invalid ConventionID: no matching convention found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF NOT EXISTS (SELECT 1 FROM Users WHERE UserID = @UserID)
        BEGIN
            RAISERROR('Invalid UserID: no matching user found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        DECLARE @CurSubject INT = (SELECT SubjectID FROM Convention WHERE ConventionID =
            @ConventionID);
        DECLARE @ConvStart DATE = (SELECT StartDate FROM Convention WHERE ConventionID =
            @ConventionID);
        DECLARE @ConvEnd DATE = (SELECT DATEADD(DAY, Duration, StartDate) FROM
            Convention WHERE ConventionID = @ConventionID);

        DECLARE @ClassID INT;
        DECLARE Stationary_class_cursor CURSOR FOR
```

```
        SELECT MeetingID
        FROM ClassMeeting
        JOIN StationaryClass ON ClassMeeting.ClassMeetingID = StationaryClass.MeetingID
        WHERE ClassMeeting.SubjectID = @CurSubject
          AND StationaryClass.StartDate BETWEEN @ConvStart AND @ConvEnd;

        OPEN Stationary_class_cursor;
        FETCH NEXT FROM Stationary_class_cursor INTO @ClassID;
        WHILE @@FETCH_STATUS = 0
        BEGIN
            EXEC p_EnrollStudentInSyncClassMeeting @UserID, @ClassID;
            FETCH NEXT FROM Stationary_class_cursor INTO @ClassID;
        END;
        CLOSE Stationary_class_cursor;
        DEALLOCATE Stationary_class_cursor;

        DECLARE @AsyncClassID INT;
        DECLARE Async_class_cursor CURSOR FOR
        SELECT MeetingID
        FROM ClassMeeting
        JOIN OfflineVideoClass ON ClassMeeting.ClassMeetingID =
            OfflineVideoClass.MeetingID
        WHERE ClassMeeting.SubjectID = @CurSubject
          AND OfflineVideoClass.StartDate BETWEEN @ConvStart AND @ConvEnd;

        OPEN Async_class_cursor;
        FETCH NEXT FROM Async_class_cursor INTO @AsyncClassID;
        WHILE @@FETCH_STATUS = 0
        BEGIN
            EXEC p_EnrollStudentInAsyncClass @UserID, @AsyncClassID;
            FETCH NEXT FROM Async_class_cursor INTO @AsyncClassID;
        END;
        CLOSE Async_class_cursor;
        DEALLOCATE Async_class_cursor;

        DECLARE @OnlineClassID INT;
        DECLARE Online_class_cursor CURSOR FOR
        SELECT MeetingID
        FROM ClassMeeting
        JOIN OnlineLiveClass ON ClassMeeting.ClassMeetingID = OnlineLiveClass.MeetingID
        WHERE ClassMeeting.SubjectID = @CurSubject
          AND OnlineLiveClass.StartDate BETWEEN @ConvStart AND @ConvEnd;

        OPEN Online_class_cursor;
        FETCH NEXT FROM Online_class_cursor INTO @OnlineClassID;
        WHILE @@FETCH_STATUS = 0
        BEGIN
            EXEC p_EnrollStudentInSyncClassMeeting @UserID, @OnlineClassID;
            FETCH NEXT FROM Online_class_cursor INTO @OnlineClassID;
        END;
        CLOSE Online_class_cursor;
        DEALLOCATE Online_class_cursor;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.39 Procedura p_EnrollStudentInSyncClassMeeting - Michał Szymocha

Procedura p_EnrollStudentInSyncClassMeeting zapisuje studenta na określone synchroniczne spotkanie klasowe.

Walidacje:

- Sprawdza, czy @MeetingID istnieje w tabeli ClassMeeting jako spotkanie synchroniczne.
- Weryfikuje, czy @StudentID istnieje w tabeli Users jako student.
- Sprawdza, czy student nie jest już zapisany na to spotkanie.

Jeśli walidacje są spełnione, procedura dodaje wpis do tabeli SyncClassEnrollments. W przeciwnym razie transakcja jest wycofywana.

```
CREATE OR ALTER Procedure p_EnrollStudentInSyncClassMeeting
(
    @UserID INT,
    @ClassID INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM ClassMeeting WHERE ClassMeetingID = @ClassID)
        BEGIN
            RAISERROR('Invalid ClassID: no matching class found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF NOT EXISTS (SELECT 1 FROM Users WHERE UserID = @UserID)
        BEGIN
            RAISERROR('Invalid UserID: no matching user found.', 16, 2);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF EXISTS (SELECT 1 FROM SyncClassDetails WHERE StudentID = @UserID AND
            MeetingID = @ClassID)
        BEGIN
            RAISERROR('User is already enrolled in this class.', 16, 3);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF (Select MeetingType from ClassMeeting where ClassMeetingID = @ClassID) in
            ('Stationary', 'OnlineLiveClass')
        BEGIN
            EXEC p_EnrollStudentInSyncClassMeeting @UserID, @ClassID;
        END;

        ELSE
        BEGIN
            EXEC p_EnrollStudentInAsyncClass @UserID, @ClassID;
        END;


        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
```

```
    END CATCH;
END;
```

6.0.40 Procedura p_EnrollStudentInAsyncClass - Michał Szymocha

Procedura p_EnrollStudentInAsyncClass zapisuje studenta na określone asynchroniczne zajęcia klasowe.
Walidacje:

- Sprawdza, czy @ClassID istnieje w tabeli AsyncClass.
- Weryfikuje, czy @StudentID istnieje w tabeli Users jako student.
- Sprawdza, czy student nie jest już zapisany na te zajęcia.

Jeśli walidacje są spełnione, procedura dodaje wpis do tabeli AsyncClassEnrollments. W przeciwnym razie transakcja jest wycofywana.

```
Create or alter procedure p_EnrollStudentInAsyncClass
(
    @UserID INT,
    @ClassID INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM ClassMeeting WHERE ClassMeetingID = @ClassID)
        BEGIN
            RAISERROR('Invalid ClassID: no matching class found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF NOT EXISTS (SELECT 1 FROM Users WHERE UserID = @UserID)
        BEGIN
            RAISERROR('Invalid UserID: no matching user found.', 16, 2);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF EXISTS (SELECT 1 FROM AsyncClassDetails WHERE StudentID = @UserID AND
            MeetingID = @ClassID)
        BEGIN
            RAISERROR('User is already enrolled in this class.', 16, 3);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        INSERT INTO AsyncClassDetails (StudentID, MeetingID)
        VALUES (@UserID, @ClassID);

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.41 Procedura p_EnrollStudentInSubject - Michał Szymocha

Procedura p_EnrollStudentInSubject zapisuje studenta na określony przedmiot.
Walidacje:

- Sprawdza, czy @SubjectID istnieje w tabeli Subject. -Weryfikuje, czy @StudentID istnieje w tabeli Users jako student.
- Sprawdza, czy student nie jest już zapisany na ten przedmiot.

Jeśli walidacje są spełnione, procedura dodaje wpis do tabeli SubjectEnrollments. W przeciwnym razie transakcja jest wycofywana.

```
CREATE or ALTER PROCEDURE p_EnrollStudentInSubject
(
    @UserID INT,
    @SubjectID INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM Subject WHERE SubjectID = @SubjectID)
        BEGIN
            RAISERROR('Invalid SubjectID: no matching subject found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF NOT EXISTS (SELECT 1 FROM Users WHERE UserID = @UserID)
        BEGIN
            RAISERROR('Invalid UserID: no matching user found.', 16, 2);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF EXISTS (SELECT 1 FROM SubjectDetails WHERE StudentID = @UserID AND SubjectID
            = @SubjectID)
        BEGIN
            RAISERROR('User is already enrolled in this subject.', 16, 3);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        INSERT INTO SubjectDetails (StudentID, SubjectID)
        VALUES (@UserID, @SubjectID);

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.42 Procedura p_StudiesSyllabus - Michał Szymocha

Procedura wyświetla syllabus studiów z podziałem na semestry.
Walidacja:

- Sprawdza czy istnieją studia o StudiesID równym podanym @StudiesID.

```sql
CREATE or alter PROCEDURE p_StudiesSyllabus
    @StudiesID int
AS
BEGIN
    IF NOT EXISTS (SELECT 1
    FROM Studies
    WHERE StudiesID = @StudiesID)
    BEGIN
        RAISERROR ('StudiesID does not exist.', 16, 1)
        RETURN
    END;

    WITH
        SemesterNumbers
        AS
        (
            SELECT
                sd.SemesterID,
                sd.StudiesID,
                DENSE_RANK() OVER (PARTITION BY sd.StudiesID ORDER BY sd.SemesterID) AS
                    SemesterNumber
            FROM SemesterDetails sd
            WHERE sd.StudiesID = @StudiesID
        )
    SELECT DISTINCT
        sn.SemesterNumber,
        s.SubjectID,
        s.SubjectName,
        s.SubjectDescription
    FROM SemesterNumbers sn
        INNER JOIN Convention c ON sn.SemesterID = c.SemesterID
        INNER JOIN Subject s ON c.SubjectID = s.SubjectID
    ORDER BY sn.SemesterNumber, s.SubjectName;
END
```

6.0.43 Procedura p_ChangeAttendanceStatus

Procedura umożliwia zaznaczenie obecności studenta na spotkaniu.
Walidacja:

- Sprawdza czy istnieje student o określonym UserID.
- Sprawdza czy istenieje spotkanie o określonym ClassID.

```sql
CREATE or ALTER Procedure p_ChangeAttendanceStatus
(
    @UserID INT,
    @ClassID INT,
    @Attendance BIT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM ClassMeeting WHERE ClassMeetingID = @ClassID)
        BEGIN
            RAISERROR('Invalid ClassID: no matching class found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

```

```
        IF NOT EXISTS (SELECT 1 FROM Users WHERE UserID = @UserID)
        BEGIN
            RAISERROR('Invalid UserID: no matching user found.', 16, 2);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF NOT EXISTS (SELECT 1 FROM SyncClassDetails WHERE StudentID = @UserID AND
            MeetingID = @ClassID)
        BEGIN
            RAISERROR('User is not enrolled in this class.', 16, 3);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        UPDATE SyncClassDetails
        SET Attendance = @Attendance
        WHERE StudentID = @UserID AND MeetingID = @ClassID;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.44 Procedura p_ChangeViewStatusAsyncClass

Procedura umożliwia zaznaczenie daty obejrzenia materiału VOD przez studenta.
Walidacja:

- Sprawdza czy istnieje student o określonym UserID.
- Sprawdza czy istnieje spotkanie VOD o określonym ClassID.

```
CREATE or ALTER Procedure p_ChangeViewStatusAsyncClass
(
    @UserID INT,
    @ClassID INT,
    @Viewed DATETIME
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM ClassMeeting WHERE ClassMeetingID = @ClassID)
        BEGIN
            RAISERROR('Invalid ClassID: no matching class found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF NOT EXISTS (SELECT 1 FROM Users WHERE UserID = @UserID)
        BEGIN
            RAISERROR('Invalid UserID: no matching user found.', 16, 2);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

```

```
        IF NOT EXISTS (SELECT 1 FROM AsyncClassDetails WHERE StudentID = @UserID AND
            MeetingID = @ClassID)
        BEGIN
            RAISERROR('User is not enrolled in this class.', 16, 3);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        UPDATE AsyncClassDetails
        SET ViewDate = @Viewed
        WHERE StudentID = @UserID AND MeetingID = @ClassID;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.45 Procedura p_StudiesSemesterSchedule

Procedura zwraca semstralny plan zajęć dla danych studiów. Należy podać ID studiów oraz numer (nie ID) semsestru, czyli np. wywalołanie tej procedury z argumentami 5,3 zwróci plan zajęć dla trzeciego semestru studiów o ID 5.

Walidacja:

- Sprawdza czy istnieją studia o ID = @StudiesID.
- Sprawdza czy istnieje taki semestr dla danych studiów.

```
CREATE or Alter PROCEDURE p_StudiesSemesterSchedule
    @StudiesID int,
    @Semester int
AS
BEGIN
    IF NOT EXISTS (SELECT 1 FROM Studies WHERE StudiesID = @StudiesID)
    BEGIN
        RAISERROR('StudiesID does not exist.', 16, 1);
        RETURN;
    END;

    Declare @SemesterID INT;
    SET @SemesterID = (SELECT MIN(SemesterID) from SemesterDetails where StudiesID =
        @StudiesID) + @Semester - 1;
    DECLARE @SemesterStartDate DATE;
    DECLARE @SemesterEndDate DATE;
    SELECT @SemesterStartDate = StartDate, @SemesterEndDate = EndDate FROM
        SemesterDetails WHERE SemesterID = @SemesterID;

    IF NOT EXISTS (SELECT 1 FROM SemesterDetails WHERE SemesterID = @SemesterID and
        StudiesID = @StudiesID)
    BEGIN
        RAISERROR('Semester does not exist for the specified StudiesID.', 16, 2);
        RETURN;
    END;

    WITH SemesterConventions AS (
        SELECT
            c.SubjectID,
            c.StartDate as ConventionStartDate,
            c.Duration as ConventionDuration,
            c.ServiceID
```

```
        FROM Convention c
        JOIN SemesterDetails sd ON sd.SemesterID = @SemesterID
        WHERE c.SemesterID = @SemesterID
    ),
    AllMeetings AS (
        SELECT
            cm.ClassMeetingID as MeetingID,
            CASE
                WHEN sc.StartDate IS NOT NULL THEN sc.StartDate
                WHEN olc.StartDate IS NOT NULL THEN olc.StartDate
                WHEN ovc.StartDate IS NOT NULL THEN ovc.StartDate
                ELSE NULL
            END as StartDate,
            CASE
                WHEN sc.StartDate IS NOT NULL THEN DATEADD(minute, DATEDIFF(minute,
                    '00:00:00', sc.Duration), sc.StartDate)
                WHEN olc.StartDate IS NOT NULL THEN DATEADD(minute, DATEDIFF(minute,
                    '00:00:00', olc.Duration), olc.StartDate)
                WHEN ovc.Deadline IS NOT NULL THEN ovc.Deadline
                ELSE NULL
            END as EndDate,
            cm.MeetingType,
            SubjectName
        FROM ClassMeeting cm
        JOIN SemesterConventions sc_conv ON cm.SubjectID = sc_conv.SubjectID
        JOIN Subject on cm.SubjectID = Subject.SubjectID
        LEFT JOIN StationaryClass sc ON cm.ClassMeetingID = sc.MeetingID
        LEFT JOIN OnlineLiveClass olc ON cm.ClassMeetingID = olc.MeetingID
        LEFT JOIN OfflineVideoClass ovc ON cm.ClassMeetingID = ovc.MeetingID
    )

    SELECT
        MeetingID,
        StartDate,
        EndDate as Deadline,
        MeetingType,
        SubjectName
    FROM AllMeetings
    where StartDate is not null and StartDate >= @SemesterStartDate and EndDate <=
        @SemesterEndDate
    ORDER BY StartDate, MeetingID;
END
```

6.0.46 Procedura p_EditGrade - Michał Szymocha

Procedura p_EditGrade umożliwia edytowanie istniejącej oceny dla studenta.
Walidacje:

- Sprawdza, czy @GradeID istnieje w tabeli Grades.
- Weryfikuje, czy nowa wartość @GradeValue mieści się w dozwolonym zakresie.

Jeśli walidacje są spełnione, procedura aktualizuje ocenę w tabeli Grades. W przeciwnym razie transakcja jest wycofywana.

```
CREATE OR ALTER PROCEDURE p_EditGrade
(
    @GradeID INT,
    @GradeName VARCHAR(50) = NULL,
    @GradeValue INT = NULL
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
```

```
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM Grades WHERE GradeID = @GradeID)
        BEGIN
            RAISERROR('Invalid GradeID: no matching Grade found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF @GradeName IS NOT NULL
        BEGIN
            IF @GradeName = ''
            BEGIN
                RAISERROR('GradeName cannot be empty.', 16, 2);
                ROLLBACK TRANSACTION;
                RETURN;
            END;
            UPDATE Grades
            SET GradeName = @GradeName
            WHERE GradeID = @GradeID;
        END;

        IF @GradeValue IS NOT NULL
        BEGIN
            IF @GradeValue < 0
            BEGIN
                RAISERROR('GradeValue cannot be negative.', 16, 3);
                ROLLBACK TRANSACTION;
                RETURN;
            END;
            UPDATE Grades
            SET GradeValue = @GradeValue
            WHERE GradeID = @GradeID;
        END;

        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.47 Procedura p_DeleteGrade - Michał Szymocha

Procedura p_DeleteGrade usuwa określoną ocenę studenta.
Walidacje:

- Sprawdza, czy @GradeID istnieje w tabeli Grades.

Jeśli walidacje są spełnione, procedura usuwa wpis z tabeli Grades. W przeciwnym razie transakcja jest wycofywana.

```
CREATE OR ALTER PROCEDURE p_DeleteGrade
(
    @GradeID INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
```

```
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM Grades WHERE GradeID = @GradeID)
        BEGIN
            RAISERROR('Invalid GradeID: no matching Grade found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        DELETE FROM Grades
        WHERE GradeID = @GradeID;

        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.48 Procedura p_EditOrder - Michał Szymocha

Procedura p_EditOrder umożliwia edytowanie szczegółów zamówienia.
Walidacje:

- Sprawdza, czy @OrderID istnieje w tabeli Orders.
- Weryfikuje, czy nowe wartości, takie jak @OrderStatus czy @TotalAmount, są poprawne.

Jeśli walidacje są spełnione, procedura aktualizuje dane zamówienia w tabeli Orders. W przeciwnym razie transakcja jest wycofywana.

```
CREATE OR ALTER PROCEDURE p_EditOrder
(
    @OrderID INT,
    @PaymentLink VARCHAR(60) = NULL,
    @OrderDate DATETIME = NULL
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM Orders WHERE OrderID = @OrderID)
        BEGIN
            RAISERROR('Invalid OrderID: no matching order found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF @OrderDate IS NOT NULL AND @OrderDate > GETDATE()
        BEGIN
            RAISERROR('OrderDate cannot be in the future.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        UPDATE Orders
        SET
            PaymentLink = CASE WHEN @PaymentLink IS NOT NULL THEN @PaymentLink ELSE
                PaymentLink END,
```

```
            OrderDate = CASE WHEN @OrderDate IS NOT NULL THEN @OrderDate ELSE OrderDate
                END
        WHERE OrderID = @OrderID;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.49 Procedura p_AddOrder - Michał Szymocha

Procedura p_AddOrder umożliwia dodanie nowego zamówienia do systemu.
Walidacje:

- Sprawdza, czy @CustomerID istnieje w tabeli Users jako klient.
- Weryfikuje, czy wartość @TotalAmount jest większa od 0.

Jeśli walidacje są spełnione, procedura dodaje nowe zamówienie do tabeli Orders. W przeciwnym razie transakcja jest wycofywana.

```
CREATE OR ALTER PROCEDURE p_AddOrder
(
    @OrderID INT,
    @UserID INT,
    @OrderDate DATETIME = NULL,
    @PaymentLink VARCHAR(60) = NULL
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;


        IF NOT EXISTS (SELECT 1 FROM ServiceUserDetails WHERE ServiceUserID = @UserID)
        BEGIN
            RAISERROR('Invalid UserID: no matching user found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF @OrderDate > GETDATE()
        BEGIN
            RAISERROR('OrderDate cannot be in the future.', 16, 2);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        INSERT INTO Orders (OrderID, UserID, OrderDate, PaymentLink)
        VALUES (@OrderID, @UserID, @OrderDate, @PaymentLink);

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
```

```
END;
```

6.0.50 Procedura p_AddClassMeetingService - Emil Żychowicz

Procedura p_AddClassMeetingService umożliwia dodanie nowej usługi spotkania klasowego do tabeli ClassMeetingService.
Walidacja:

- Sprawdza, czy podany @ServiceID istnieje w tabeli Services.
- Sprawdza, czy cena dla studentów (@PriceStudents) jest większa od 0.

Jeśli dane wejściowe są poprawne, rekord zostaje dodany do tabeli. W przypadku błędów transakcja jest wycofywana, a użytkownik otrzymuje komunikat o błędzie.

```
CREATE OR ALTER PROCEDURE p_AddClassMeetingService
(
    @ServiceID INT,
    @PriceStudents MONEY,
    @PriceOthers MONEY = NULL
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM Services WHERE ServiceID = @ServiceID)
        BEGIN
            RAISERROR('Invalid ServiceID: no matching service found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF @PriceStudents <= 0
        BEGIN
            RAISERROR('PriceStudents must be greater than zero.', 16, 2);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        INSERT INTO ClassMeetingService (ServiceID, PriceStudents, PriceOthers)
        VALUES (@ServiceID, @PriceStudents, @PriceOthers);

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.51 Procedura p_AddConventionService - Emil Żychowicz

Procedura p_AddConventionService umożliwia dodanie usługi konwencyjnej do tabeli ConventionService.
Weryfikuje poprawność danych wejściowych:

- Sprawdza, czy @ServiceID istnieje w tabeli Services.
- Sprawdza, czy cena (@Price) jest większa od 0.

Po pozytywnej weryfikacji danych, procedura dodaje rekord do tabeli. W przypadku błędów transakcja jest wycofywana, a użytkownik otrzymuje odpowiedni komunikat o błędzie.

```sql
CREATE OR ALTER PROCEDURE p_AddConventionService
(
    @ServiceID INT,
    @Price MONEY
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM Services WHERE ServiceID = @ServiceID)
        BEGIN
            RAISERROR('Invalid ServiceID: no matching service found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF @Price <= 0
        BEGIN
            RAISERROR('Price must be greater than zero.', 16, 2);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        INSERT INTO ConventionService (ServiceID, Price)
        VALUES (@ServiceID, @Price);

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.52 Procedura p_AddCourseParticipant - Emil Żychowicz

Procedura p_AddCourseParticipant umożliwia dodanie uczestnika do kursu. Weryfikuje dane wejściowe:

- Sprawdza, czy @ParticipantID istnieje w tabeli ServiceUserDetails.
- Sprawdza, czy @CourseID istnieje w tabeli Courses.
- Sprawdza, czy uczestnik nie został już zapisany na ten kurs.

Po pomyślnym dodaniu uczestnika, procedura iteruje przez spotkania powiązane z kursem (z tabeli COURSE_INFO) i przypisuje uczestnika do odpowiednich szczegółów spotkań (np. stacjonarnych, online itp.). Jeśli wystąpi błąd, transakcja jest wycofywana, a użytkownik otrzymuje komunikat.

```sql
CREATE OR ALTER PROCEDURE p_AddCourseParticipant
(
    @ParticipantID INT,
    @CourseID      INT
)
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @MeetingID INT;
    DECLARE @MeetingType VARCHAR(20);

    -- Kursor dla przechodzenia po widoku COURSE_INFO
```

```
    DECLARE MeetingCursor CURSOR FOR
    SELECT MeetingID, MeetingType
    FROM COURSE_INFO
    WHERE CourseID = @CourseID;

    BEGIN TRY
        BEGIN TRANSACTION;
    DECLARE @Vacancies INT
    SET @Vacancies = (SELECT Vacancies FROM COURSE_INFO WHERE CourseID = @CourseID GROUP
       BY CourseID, Vacancies);
    IF @Vacancies <= 0
    BEGIN
      RAISERROR('No more vacancies', 16, 1);
    END
        IF NOT EXISTS (SELECT 1 FROM ServiceUserDetails WHERE ServiceUserID =
           @ParticipantID)
        BEGIN
            RAISERROR('Invalid ParticipantID: no matching participant found in
               ServiceUserDetails.', 16, 2);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF NOT EXISTS (SELECT 1 FROM Courses WHERE CourseID = @CourseID)
        BEGIN
            RAISERROR('Invalid CourseID: no matching course found.', 16, 3);
            ROLLBACK TRANSACTION;
            RETURN;
        END;


        IF EXISTS (SELECT 1 FROM CourseParticipants WHERE ParticipantID = @ParticipantID
           AND CourseID = @CourseID)
        BEGIN
            RAISERROR('The participant is already enrolled in this course.', 16, 4);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        INSERT INTO CourseParticipants
            (ParticipantID, CourseID)
        VALUES
            (@ParticipantID, @CourseID);

        OPEN MeetingCursor;

        FETCH NEXT FROM MeetingCursor INTO @MeetingID, @MeetingType;

        WHILE @@FETCH_STATUS = 0
        BEGIN
            IF @MeetingType = 'Stationary'
            BEGIN
                EXEC p_AddStationaryMeetingDetails @MeetingID, @ParticipantID;
            END
            ELSE IF @MeetingType = 'Online Live'
            BEGIN
                EXEC p_AddOnlineLiveMeetingDetails @MeetingID, @ParticipantID;
            END
            ELSE IF @MeetingType = 'Offline Video'
            BEGIN
                EXEC p_AddOfflineVideoDetails @MeetingID, @ParticipantID;
            END
            ELSE
            BEGIN
                RAISERROR('Invalid MeetingType: %s', 16, 4, @MeetingType);
```

```
                ROLLBACK TRANSACTION;
                CLOSE MeetingCursor;
                DEALLOCATE MeetingCursor;
                RETURN;
            END;

            FETCH NEXT FROM MeetingCursor INTO @MeetingID, @MeetingType;
        END;

        CLOSE MeetingCursor;
        DEALLOCATE MeetingCursor;

        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        IF CURSOR_STATUS('global', 'MeetingCursor') >= 0
        BEGIN
            CLOSE MeetingCursor;
            DEALLOCATE MeetingCursor;
        END;

        THROW;
    END CATCH;
END;
GO
```

6.0.53 Procedura p_AddCourseService - Emil Żychowicz

Procedura p_AddCourseService pozwala dodać nową usługę kursową do tabeli CourseService.
Weryfikuje dane wejściowe:

- Sprawdza, czy @ServiceID istnieje w tabeli Services.
- Sprawdza, czy zaliczka (@AdvanceValue) jest większa od 0.
- Sprawdza, czy pełna cena (@FullPrice) jest większa lub równa zaliczce.

Jeśli wszystkie warunki są spełnione, rekord zostaje dodany do tabeli. W przypadku błędów procedura wycofuje transakcję i generuje odpowiedni komunikat.

```
CREATE OR ALTER PROCEDURE p_AddCourseService
(
    @ServiceID INT,
    @AdvanceValue MONEY,
    @FullPrice MONEY
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM Services WHERE ServiceID = @ServiceID)
        BEGIN
            RAISERROR('Invalid ServiceID: no matching service found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF @AdvanceValue <= 0
        BEGIN
```

```
            RAISERROR('AdvanceValue must be greater than zero.', 16, 2);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF @FullPrice < @AdvanceValue
        BEGIN
            RAISERROR('FullPrice must be greater than or equal to AdvanceValue.', 16, 3);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        INSERT INTO CourseService (ServiceID, AdvanceValue, FullPrice)
        VALUES (@ServiceID, @AdvanceValue, @FullPrice);

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.54 Procedura p_AddOfflineVideo - Emil Żychowicz

Procedura p_AddOfflineVideo umożliwia dodanie nowego wideo offline do tabeli OfflineVideo.
Weryfikuje dane wejściowe:

- Sprawdza, czy link do wideo (@VideoLink) nie jest pusty.
- Sprawdza, czy @ModuleID istnieje w tabeli Modules.
- Sprawdza, czy czas trwania wideo (@VideoDuration) jest większy od 00:00:00.
- Sprawdza, czy @TeacherID istnieje w tabeli Employees.

Jeśli wszystkie warunki są spełnione, procedura generuje unikalny MeetingID i dodaje rekord do tabeli OfflineVideo. W przypadku błędów transakcja jest wycofywana.

```
CREATE OR ALTER PROCEDURE p_AddOfflineVideo
(
    @VideoLink        VARCHAR(60),
    @ModuleID         INT,
    @VideoDuration    TIME = '01:30:00',
    @TeacherID        INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF @VideoLink IS NULL OR LEN(@VideoLink) = 0
        BEGIN
            RAISERROR('Invalid VideoLink: it cannot be NULL or empty.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF NOT EXISTS (SELECT 1 FROM Modules WHERE ModuleID = @ModuleID)
        BEGIN
            RAISERROR('Invalid ModuleID: no matching Module found.', 16, 2);
            ROLLBACK TRANSACTION;
            RETURN;
```

```
        END;

        IF @VideoDuration IS NULL OR @VideoDuration <= '00:00:00'
        BEGIN
            RAISERROR('Invalid VideoDuration: duration must be greater than 00:00:00.',
                16, 3);
            ROLLBACK TRANSACTION;
            RETURN;
        END;


        IF NOT EXISTS (SELECT 1 FROM Employees WHERE EmployeeID = @TeacherID)
        BEGIN
            RAISERROR('Invalid TeacherID: no matching Employee found.', 16, 4);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        DECLARE @NextMeetingID INT;
        IF NOT EXISTS (SELECT 1 FROM OfflineVideo)
        BEGIN
            SET @NextMeetingID = 1;
        END
        ELSE
        BEGIN
            SELECT @NextMeetingID = MAX(MeetingID) + 1 FROM OfflineVideo;
        END;

        INSERT INTO OfflineVideo
            (MeetingID, VideoLink, ModuleID, VideoDuration, TeacherID)
        VALUES
            (@NextMeetingID, @VideoLink, @ModuleID, @VideoDuration, @TeacherID);

        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
GO
```

6.0.55 Procedura p_AddOfflineVideoDetails - Emil Żychowicz

Procedura p_AddOfflineVideoDetails umożliwia przypisanie szczegółów uczestnictwa w wideo offline do tabeli OfflineVideoDetails.

Weryfikuje dane wejściowe:

- Sprawdza, czy @MeetingID istnieje w tabeli OfflineVideo.
- Sprawdza, czy @ParticipantID istnieje w tabeli ServiceUserDetails.
- Sprawdza, czy uczestnik nie został już dodany do tego samego wideo.

Jeśli dane wejściowe są poprawne, rekord jest dodawany do tabeli. W przeciwnym razie transakcja jest wycofywana, a użytkownik otrzymuje komunikat o błędzie.

```
CREATE OR ALTER PROCEDURE p_AddOfflineVideoDetails
(
    @MeetingID     INT,
    @ParticipantID INT,
    @DateOfViewing DATE = NULL
)
```

```
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM OfflineVideo WHERE MeetingID = @MeetingID)
        BEGIN
            RAISERROR('Invalid MeetingID: no matching offline video found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF NOT EXISTS (SELECT 1 FROM ServiceUserDetails WHERE ServiceUserID =
            @ParticipantID)
        BEGIN
            RAISERROR('Invalid ParticipantID: no matching participant found.', 16, 2);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF EXISTS (SELECT 1 FROM OfflineVideoDetails WHERE MeetingID = @MeetingID AND
            ParticipantID = @ParticipantID)
        BEGIN
            RAISERROR('The participant is already added to this offline video.', 16, 3);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        INSERT INTO OfflineVideoDetails
            (MeetingID, ParticipantID, DateOfViewing)
        VALUES
            (@MeetingID, @ParticipantID, @DateOfViewing);

        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH;
END;
GO
```

6.0.56 Procedura p_AddOnlineLiveMeeting - Emil Żychowicz

Procedura p_AddOnlineLiveMeeting umożliwia dodanie nowego spotkania online na żywo do tabeli OnlineLiveMeeting.

Weryfikuje poprawność danych wejściowych:

- Sprawdza, czy data spotkania (@MeetingDate) mieści się w zakresie od 2015-01-01 do 2030-01-01.
- Sprawdza, czy czas trwania spotkania (@MeetingDuration) jest większy od 00:00:00.
- Sprawdza, czy @ModuleID istnieje w tabeli Modules.
- Sprawdza, czy @TeacherID istnieje w tabeli Employees.
- Wymaga, aby @PlatformName lub @Link zostały podane (przynajmniej jeden z nich).

Po pomyślnej weryfikacji danych, procedura generuje unikalny MeetingID i dodaje rekord do tabeli. W przypadku błędów transakcja jest wycofywana.

```
CREATE OR ALTER PROCEDURE p_AddOnlineLiveMeeting
(
    @PlatformName      VARCHAR(20),
```

```
    @Link               VARCHAR(60),
    @VideoLink          VARCHAR(60),
    @ModuleID           INT,
    @MeetingDate        DATETIME,
    @MeetingDuration    TIME = '01:30:00',
    @TeacherID          INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF @MeetingDate IS NULL OR @MeetingDate <= '2015-01-01' OR @MeetingDate >= '2030-01-01'
        BEGIN
            RAISERROR('Invalid MeetingDate: it must be between 2015-01-01 and 2030-01-01.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF @MeetingDuration IS NULL OR @MeetingDuration <= '00:00:00'
        BEGIN
            RAISERROR('Invalid MeetingDuration: duration must be greater than 00:00:00.', 16, 2);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF NOT EXISTS (SELECT 1 FROM Modules WHERE ModuleID = @ModuleID)
        BEGIN
            RAISERROR('Invalid ModuleID: no matching Module found.', 16, 3);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF NOT EXISTS (SELECT 1 FROM Employees WHERE EmployeeID = @TeacherID)
        BEGIN
            RAISERROR('Invalid TeacherID: no matching Employee found.', 16, 4);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF (@PlatformName IS NULL OR LEN(@PlatformName) = 0) AND (@Link IS NULL OR LEN(@Link) = 0)
        BEGIN
            RAISERROR('Invalid PlatformName and Link: at least one must be provided.', 16, 5);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        DECLARE @NextMeetingID INT;
        IF NOT EXISTS (SELECT 1 FROM OnlineLiveMeeting)
        BEGIN
            SET @NextMeetingID = 1;
        END
        ELSE
        BEGIN
            SELECT @NextMeetingID = MAX(MeetingID) + 1 FROM OnlineLiveMeeting;
        END;

        INSERT INTO OnlineLiveMeeting
```

```
            (MeetingID, PlatformName, Link, VideoLink, ModuleID, MeetingDate,
                MeetingDuration, TeacherID)
        VALUES
            (@NextMeetingID, @PlatformName, @Link, @VideoLink, @ModuleID, @MeetingDate,
                @MeetingDuration, @TeacherID);

        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
GO
```

6.0.57 Procedura p_AddOnlineLiveMeetingDetails - Emil Żychowicz

Procedura p_AddOnlineLiveMeetingDetails umożliwia dodanie szczegółów spotkania online dla konkretnego uczestnika do tabeli OnlineLiveMeetingDetails.

Weryfikuje poprawność danych wejściowych:

- Sprawdza, czy podany @MeetingID istnieje w tabeli OnlineLiveMeeting.
- Sprawdza, czy podany @ParticipantID istnieje w tabeli ServiceUserDetails.
- Weryfikuje, czy uczestnik nie został już dodany do tego spotkania.

Jeśli dane wejściowe są poprawne, rekord zostaje dodany do tabeli. W przypadku błędów transakcja jest wycofywana, a użytkownik otrzymuje komunikat o błędzie.

```
CREATE OR ALTER PROCEDURE p_AddOnlineLiveMeetingDetails
(
    @MeetingID      INT,
    @ParticipantID INT,
    @Attendance     BIT = NULL
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM OnlineLiveMeeting WHERE MeetingID = @MeetingID)
        BEGIN
            RAISERROR('Invalid MeetingID: no matching online live meeting found.', 16,
                1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF NOT EXISTS (SELECT 1 FROM ServiceUserDetails WHERE ServiceUserID =
            @ParticipantID)
        BEGIN
            RAISERROR('Invalid ParticipantID: no matching participant found.', 16, 2);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF EXISTS (SELECT 1 FROM OnlineLiveMeetingDetails WHERE MeetingID = @MeetingID
            AND ParticipantID = @ParticipantID)
        BEGIN
            RAISERROR('The participant is already added to this online live meeting.',
                16, 3);
```

```
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        INSERT INTO OnlineLiveMeetingDetails
            (MeetingID, ParticipantID, Attendance)
        VALUES
            (@MeetingID, @ParticipantID, @Attendance);

        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH;
END;
GO
```

6.0.58 Procedura p_CreateOrder - Emil Żychowicz

Procedura p_CreateOrder umożliwia utworzenie nowego zamówienia w tabeli Orders.
Weryfikuje poprawność danych wejściowych:

- Sprawdza, czy podany @UserID istnieje w tabeli ServiceUserDetails.
- Sprawdza, czy podana data zamówienia @OrderDate nie jest w przyszłości.

Jeśli dane wejściowe są poprawne, procedura generuje nowy identyfikator zamówienia (OrderID) i dodaje rekord do tabeli Orders. W przypadku błędów transakcja jest wycofywana, a użytkownik otrzymuje komunikat o błędzie.

```
CREATE OR ALTER PROCEDURE p_CreateOrder
(
    @UserID INT,
    @OrderDate DATETIME = NULL,
    @PaymentLink VARCHAR(60) = NULL,
  @OrderID INT OUTPUT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM ServiceUserDetails WHERE ServiceUserID = @UserID)
        BEGIN
            RAISERROR('Invalid UserID: no matching user found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF @OrderDate > GETDATE()
        BEGIN
            RAISERROR('OrderDate cannot be in the future.', 16, 2);
            ROLLBACK TRANSACTION;
            RETURN;
        END;
        DECLARE @NextOrderID INT;
        IF NOT EXISTS (SELECT 1 FROM Orders)
        BEGIN
            SET @NextOrderID = 1;
        END
```

```
        ELSE
        BEGIN
            SELECT @NextOrderID = MAX(OrderID) + 1 FROM Orders;
        END;

    BEGIN
      SET @OrderID = @NextOrderID;
    END;

        INSERT INTO Orders (OrderID, UserID, OrderDate, PaymentLink)
        VALUES (@NextOrderID, @UserID, @OrderDate, @PaymentLink);

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.59 Procedura p_AddToOrder - Emil Żychowicz

Procedura p_AddToOrder umożliwia dodanie szczegółów zamówienia, przypisując usługę do zamówienia w tabeli OrderDetails.

Weryfikuje poprawność danych wejściowych:

- Sprawdza, czy podany @OrderID istnieje w tabeli Orders.
- Sprawdza, czy podany @ServiceID istnieje w tabeli Services.
- Weryfikuje, czy kombinacja OrderID i ServiceID nie istnieje już w tabeli OrderDetails.

Jeśli dane wejściowe są poprawne, rekord zostaje dodany do tabeli. W przypadku błędów transakcja jest wycofywana, a użytkownik otrzymuje komunikat o błędzie.

```
CREATE OR ALTER PROCEDURE p_AddOrderDetail
(
    @OrderID INT,
    @ServiceID INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM Orders WHERE OrderID = @OrderID)
        BEGIN
            RAISERROR('Invalid OrderID: no matching order found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF NOT EXISTS (SELECT 1 FROM Services WHERE ServiceID = @ServiceID)
        BEGIN
            RAISERROR('Invalid ServiceID: no matching service found.', 16, 2);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF EXISTS (SELECT 1 FROM OrderDetails WHERE OrderID = @OrderID AND ServiceID =
            @ServiceID)
        BEGIN
```

```
            RAISERROR('This combination of OrderID and ServiceID already exists in
                OrderDetails.', 16, 3);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        INSERT INTO OrderDetails (OrderID, ServiceID, PrincipalAgreement)
        VALUES (@OrderID, @ServiceID, 0);

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.60 Procedura p_AddPayment - Emil Żychowicz

Procedura p_AddPayment umożliwia dodanie nowej płatności powiązanej z zamówieniem i usługą w tabeli Payments.

Weryfikuje poprawność danych wejściowych:

- Sprawdza, czy istnieje kombinacja OrderID i ServiceID w tabeli OrderDetails.
- Sprawdza, czy kwota płatności @PaymentValue jest większa lub równa 0.

Jeśli dane wejściowe są poprawne, rekord zostaje dodany do tabeli Payments. W przypadku błędów transakcja jest wycofywana, a użytkownik otrzymuje komunikat o błędzie.

```
CREATE OR ALTER PROCEDURE p_AddPayment
(
    @PaymentValue MONEY,
    @PaymentDate DATETIME = NULL,
    @ServiceID INT,
    @OrderID INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

    IF NOT EXISTS (SELECT 1 FROM OrderDetails WHERE OrderID = @OrderID AND ServiceID =
        @ServiceID)
        BEGIN
            RAISERROR('No matching pair (OrderID, ServiceID) in OrderDetails', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;


        IF @PaymentValue < 0
        BEGIN
            RAISERROR('PaymentValue must be greater or equal zero.', 16, 3);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

    DECLARE @PaymentID INT;
    IF NOT EXISTS (SELECT 1 FROM Payments)
    BEGIN
```

```
      SET @PaymentID = 1;
    END;
    ELSE
    BEGIN
      SELECT @PaymentID = MAX(PaymentID) + 1 FROM Payments;
    END;
        INSERT INTO Payments (PaymentID, PaymentValue, PaymentDate, ServiceID, OrderID)
        VALUES (@PaymentID, @PaymentValue, @PaymentDate, @ServiceID, @OrderID);

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.61 Procedura p_AddService - Emil Żychowicz

Procedura p_AddService umożliwia dodanie nowej usługi do tabeli Services.
Weryfikuje poprawność danych wejściowych:

- Sprawdza, czy podany typ usługi @ServiceType jest jednym z dozwolonych (ClassMeetingService, -- StudiesService, CourseService, WebinarService, ConventionService).

Jeśli dane wejściowe są poprawne, procedura generuje nowy identyfikator usługi (ServiceID) i dodaje rekord do tabeli Services. W przypadku błędów transakcja jest wycofywana, a użytkownik otrzymuje komunikat o błędzie.

```
CREATE OR ALTER PROCEDURE p_AddService
(
    @ServiceType VARCHAR(30),
  @result INT OUTPUT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF @ServiceType NOT IN ('ClassMeetingService', 'StudiesService',
            'CourseService', 'WebinarService', 'ConventionService')
        BEGIN
            RAISERROR('Invalid ServiceType: must be one of the allowed types.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;



        DECLARE @NextServiceID INT;
        IF NOT EXISTS (SELECT 1 FROM Courses)
        BEGIN
            SET @NextServiceID = 1;
        END
        ELSE
        BEGIN
            SELECT @NextServiceID = MAX(ServiceID) + 1 FROM Services;
        END;

    BEGIN
```

```
      SET @result = @NextServiceID;
    END

        INSERT INTO Services (ServiceID, ServiceType)
        VALUES (@NextServiceID, @ServiceType);

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.62 Procedura p_AddStationaryMeeting - Emil Żychowicz

Procedura p_AddStationaryMeeting umożliwia dodanie nowego spotkania stacjonarnego do tabeli StationaryMeeting.

Weryfikuje poprawność danych wejściowych:

- Sprawdza, czy data spotkania @MeetingDate jest w przyszłości.
- Sprawdza, czy czas trwania spotkania @MeetingDuration jest większy niż 00:00:00.
- Weryfikuje, czy podany moduł (@ModuleID), sala (@RoomID), i nauczyciel (@TeacherID) istnieją w odpowiednich tabelach.
- Sprawdza, czy wielkość grupy @GroupSize jest większa od 0. Jeśli dane wejściowe są poprawne, rekord zostaje dodany do tabeli. W przypadku błędów transakcja jest wycofywana, a użytkownik otrzymuje komunikat o błędzie.

```
CREATE OR ALTER PROCEDURE p_AddStationaryMeeting
(
    @MeetingDate        DATETIME,
    @MeetingDuration    TIME,
    @ModuleID           INT,
    @RoomID             INT,
    @GroupSize          INT,
    @TeacherID          INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF @MeetingDate IS NULL OR @MeetingDate < GETDATE()
        BEGIN
            RAISERROR('Invalid MeetingDate: date must be in the future.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;
        IF @MeetingDuration IS NULL OR @MeetingDuration <= '00:00:00'
        BEGIN
            RAISERROR('Invalid MeetingDuration: duration must be greater than
                00:00:00.', 16, 2);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF NOT EXISTS (SELECT 1 FROM Modules WHERE ModuleID = @ModuleID)
        BEGIN
            RAISERROR('Invalid ModuleID: no matching Module found.', 16, 3);
```

```
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF NOT EXISTS (SELECT 1 FROM Rooms WHERE RoomID = @RoomID)
        BEGIN
            RAISERROR('Invalid RoomID: no matching Room found.', 16, 4);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF NOT EXISTS (SELECT 1 FROM Employees WHERE EmployeeID = @TeacherID)
        BEGIN
            RAISERROR('Invalid TeacherID: no matching Employee found.', 16, 5);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF @GroupSize <= 0
        BEGIN
            RAISERROR('Invalid GroupSize: must be greater than 0.', 16, 6);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        DECLARE @NextMeetingID INT;
        IF NOT EXISTS (SELECT 1 FROM StationaryMeeting)
        BEGIN
            SET @NextMeetingID = 1;
        END
        ELSE
        BEGIN
            SELECT @NextMeetingID = MAX(MeetingID) + 1 FROM StationaryMeeting;
        END;

        INSERT INTO StationaryMeeting
            (MeetingID, MeetingDate, MeetingDuration, ModuleID, RoomID, GroupSize,
                TeacherID)
        VALUES
            (@NextMeetingID, @MeetingDate, @MeetingDuration, @ModuleID, @RoomID,
                @GroupSize, @TeacherID);

        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
GO
```

6.0.63 Procedura p_AddStationaryMeetingDetails - Emil Żychowicz

Procedura p_AddStationaryMeetingDetails umożliwia dodanie uczestnika do spotkania stacjonarnego z opcjonalnym statusem obecności w tabeli StationaryMeetingDetails.

Weryfikuje poprawność danych wejściowych:

- Sprawdza, czy podany @MeetingID istnieje w tabeli StationaryMeeting.
- Sprawdza, czy podany @ParticipantID istnieje w tabeli ServiceUserDetails.
- Weryfikuje, czy uczestnik nie został już dodany do spotkania.

Jeśli dane wejściowe są poprawne, rekord zostaje dodany do tabeli. W przypadku błędów transakcja jest wycofywana, a użytkownik otrzymuje komunikat o błędzie.

```
CREATE OR ALTER PROCEDURE p_AddStationaryMeetingDetails
(
    @MeetingID      INT,
    @ParticipantID INT,
    @Attendance     BIT = NULL
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM StationaryMeeting WHERE MeetingID = @MeetingID)
        BEGIN
            RAISERROR('Invalid MeetingID: no matching stationary meeting found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF NOT EXISTS (SELECT 1 FROM ServiceUserDetails WHERE ServiceUserID =
            @ParticipantID)
        BEGIN
            RAISERROR('Invalid ParticipantID: no matching participant found.', 16, 2);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF EXISTS (SELECT 1 FROM StationaryMeetingDetails WHERE MeetingID = @MeetingID
            AND ParticipantID = @ParticipantID)
        BEGIN
            RAISERROR('The participant is already added to this stationary meeting.',
                16, 3);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        INSERT INTO StationaryMeetingDetails
            (MeetingID, ParticipantID, Attendance)
        VALUES
            (@MeetingID, @ParticipantID, @Attendance);

        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH;
END;
GO
```

6.0.64 Procedura p_AddStudiesService - Emil Żychowicz

Procedura p_AddStudiesService umożliwia dodanie nowej usługi studiów do tabeli StudiesService.
Weryfikuje poprawność danych wejściowych:

- Sprawdza, czy podany @ServiceID istnieje w tabeli Services.
- Weryfikuje, czy opłata za usługę @EntryFee jest większa od 0.

Jeśli dane wejściowe są poprawne, rekord zostaje dodany do tabeli StudiesService. W przypadku błędów transakcja jest wycofywana, a użytkownik otrzymuje komunikat o błędzie.

```
CREATE OR ALTER PROCEDURE p_AddStudiesService
(
    @ServiceID INT,
    @EntryFee MONEY
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM Services WHERE ServiceID = @ServiceID)
        BEGIN
            RAISERROR('Invalid ServiceID: no matching service found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF @EntryFee <= 0
        BEGIN
            RAISERROR('EntryFee must be greater than zero.', 16, 2);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        INSERT INTO StudiesService (ServiceID, EntryFee)
        VALUES (@ServiceID, @EntryFee);

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.65 Procedura p_AddWebinarService - Emil Żychowicz

Procedura p_AddWebinarService umożliwia dodanie nowej usługi webinaru do tabeli WebinarService.
Weryfikuje poprawność danych wejściowych:

- Sprawdza, czy podany @ServiceID istnieje w tabeli Services.
- Weryfikuje, czy cena usługi @Price jest większa od 0.

Jeśli dane wejściowe są poprawne, rekord zostaje dodany do tabeli WebinarService. W przypadku błędów transakcja jest wycofywana, a użytkownik otrzymuje komunikat o błędzie.

```
CREATE OR ALTER PROCEDURE p_AddWebinarService
(
    @ServiceID INT,
    @Price MONEY
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

```

```
        IF NOT EXISTS (SELECT 1 FROM Services WHERE ServiceID = @ServiceID)
        BEGIN
            RAISERROR('Invalid ServiceID: no matching service found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF @Price <= 0
        BEGIN
            RAISERROR('Price must be greater than zero.', 16, 2);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        INSERT INTO WebinarService (ServiceID, Price)
        VALUES (@ServiceID, @Price);

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.66 Procedura p_CreateCourse - Emil Żychowicz

Procedura p_CreateCourse umożliwia utworzenie nowego kursu w tabeli Courses oraz powiązanej usługi kursowej w tabeli CourseService.

Weryfikuje poprawność danych wejściowych:

- Sprawdza, czy podany @CourseCoordinatorID istnieje w tabeli Employees.
- Weryfikuje, czy data kursu @CourseDate mieści się w przedziale od 2015-01-01 do 2030-01-01.
- Sprawdza, czy limit zapisów @EnrollmentLimit jest większy niż 1.

Jeśli dane wejściowe są poprawne, procedura dodaje nową usługę (CourseService), a następnie tworzy nowy rekord w tabeli Courses. W przypadku błędów transakcja jest wycofywana, a użytkownik otrzymuje komunikat o błędzie.

```
CREATE OR ALTER PROCEDURE p_CreateCourse
(
    @CourseName          VARCHAR(40),
    @CourseDescription   VARCHAR(255) = NULL,
    @CourseCoordinatorID INT,
    @CourseDate          DATE,
    @EnrollmentLimit     INT,
  @CoursePrice     MONEY,
  @AdvanceValue    MONEY
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM Employees WHERE EmployeeID = @CourseCoordinatorID)
        BEGIN
            RAISERROR('Invalid CourseCoordinatorID: no matching Employee found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;
```

```
24
25
26
27          IF @CourseDate <= '2015-01-01' OR @CourseDate >= '2030-01-01'
28          BEGIN
29              RAISERROR('CourseDate must be between 2015-01-01 and 2030-01-01.', 16, 3);
30              ROLLBACK TRANSACTION;
31              RETURN;
32          END;
33
34          IF @EnrollmentLimit <= 1
35          BEGIN
36              RAISERROR('EnrollmentLimit must be greater than 1.', 16, 4);
37              ROLLBACK TRANSACTION;
38              RETURN;
39          END;
40
41          DECLARE @NextCourseID INT;
42          IF NOT EXISTS (SELECT 1 FROM Courses)
43          BEGIN
44              SET @NextCourseID = 1;
45          END
46          ELSE
47          BEGIN
48              SELECT @NextCourseID = MAX(CourseID) + 1 FROM Courses;
49          END;
50
51      DECLARE @ServiceID INT;
52      EXEC p_AddService 'CourseService', @ServiceID OUTPUT;
53      EXEC p_AddCourseService @ServiceID, @AdvanceValue, @CoursePrice;
54          INSERT INTO Courses
55              (CourseID, CourseName, CourseDescription, CourseCoordinatorID, ServiceID,
                    CourseDate, EnrollmentLimit)
56          VALUES
57              (@NextCourseID, @CourseName, @CourseDescription, @CourseCoordinatorID,
                    @ServiceID, @CourseDate, @EnrollmentLimit);
58
59          COMMIT TRANSACTION;
60      END TRY
61
62      BEGIN CATCH
63          IF @@TRANCOUNT > 0
64              ROLLBACK TRANSACTION;
65          THROW;
66      END CATCH;
67  END;
68  GO
```

6.0.67 Procedura p_CreateModule - Emil Żychowicz

Procedura p_CreateModule umożliwia tworzenie nowego modułu w systemie, weryfikując poprawność danych wejściowych:

- Sprawdza, czy podany @LanguageID istnieje w tabeli Languages.
- Weryfikuje obecność @CourseID w tabeli Courses.
- Jeśli @TranslatorID nie jest NULL, sprawdza jego obecność w tabeli Employees.
- Upewnia się, że @ModuleCoordinatorID istnieje w tabeli Employees.
- Weryfikuje poprawność wartości @ModuleType (dozwolone: Offline Videos, Hybrid, Online Lives, Stationary).
- Tworzy nowy rekord w tabeli Modules, przydzielając unikalny ModuleID.

W przypadku błędów transakcja jest wycofywana, a użytkownik otrzymuje komunikat o błędzie.

```
1  CREATE OR ALTER PROCEDURE p_CreateModule
```

```
(
    @LanguageID           INT,
    @CourseID             INT,
    @TranslatorID         INT = NULL,
    @ModuleCoordinatorID  INT,
    @ModuleType           VARCHAR(30)
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM Languages WHERE LanguageID = @LanguageID)
        BEGIN
            RAISERROR('Invalid LanguageID: no matching Language found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF NOT EXISTS (SELECT 1 FROM Courses WHERE CourseID = @CourseID)
        BEGIN
            RAISERROR('Invalid CourseID: no matching Course found.', 16, 2);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF @TranslatorID IS NOT NULL AND NOT EXISTS (SELECT 1 FROM Employees WHERE
            EmployeeID = @TranslatorID)
        BEGIN
            RAISERROR('Invalid TranslatorID: no matching Employee found.', 16, 3);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF NOT EXISTS (SELECT 1 FROM Employees WHERE EmployeeID = @ModuleCoordinatorID)
        BEGIN
            RAISERROR('Invalid ModuleCoordinatorID: no matching Employee found.', 16, 4);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF LEN(@ModuleType) = 0 OR (@ModuleType != 'Offline Videos' AND @ModuleType !=
            'Hybrid' AND @ModuleType != 'Online Lives' AND
    @ModuleType != 'Stationary')
        BEGIN
            RAISERROR('Invalid ModuleType: Type must be one of those: Offline Videos,
                Hybrid, Online Lives, Stationary. Cannot be empty.', 16, 5);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        DECLARE @NextModuleID INT;
        IF NOT EXISTS (SELECT 1 FROM Modules)
        BEGIN
            SET @NextModuleID = 1;
        END
        ELSE
        BEGIN
            SELECT @NextModuleID = MAX(ModuleID) + 1 FROM Modules;
        END;

        INSERT INTO Modules
            (ModuleID, LanguageID, CourseID, TranslatorID, ModuleCoordinatorID,
```

```
                ModuleType)
        VALUES
            (@NextModuleID, @LanguageID, @CourseID, @TranslatorID, @ModuleCoordinatorID,
                @ModuleType);

        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH;
END;
GO
```

6.0.68 Procedura p_CreateOfflineVideo - Emil Żychowicz

Procedura p_CreateOfflineVideo dodaje nowy materiał wideo offline do systemu, weryfikując dane wejściowe:

- Sprawdza, czy @MeetingID nie istnieje już w tabeli OfflineVideo.
- Weryfikuje, że @VideoLink nie jest pusty.
- Sprawdza, czy podany @ModuleID istnieje w tabeli Modules.
- Weryfikuje, czy @VideoDuration jest większy od 00:00:00.
- Upewnia się, że @TeacherID istnieje w tabeli Employees.

Jeśli dane są poprawne, nowy rekord jest dodawany do tabeli OfflineVideo. W przeciwnym razie transakcja jest wycofywana, a użytkownik informowany o błędzie.

```
CREATE OR ALTER PROCEDURE p_CreateOfflineVideo
(
    @MeetingID      INT,
    @VideoLink      VARCHAR(60),
    @ModuleID       INT,
    @VideoDuration  TIME(0) = '01:30:00',
    @TeacherID      INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF EXISTS (SELECT 1 FROM OfflineVideo WHERE MeetingID = @MeetingID)
        BEGIN
            RAISERROR('MeetingID already exists in OfflineVideo.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF LEN(@VideoLink) = 0
        BEGIN
            RAISERROR('VideoLink cannot be empty.', 16, 2);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF NOT EXISTS (SELECT 1 FROM Modules WHERE ModuleID = @ModuleID)
        BEGIN
            RAISERROR('Invalid ModuleID: no matching Module found.', 16, 3);
            ROLLBACK TRANSACTION;
            RETURN;
```

```
        END;

        IF @VideoDuration <= '00:00:00'
        BEGIN
            RAISERROR('VideoDuration must be greater than 00:00:00.', 16, 4);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF NOT EXISTS (SELECT 1 FROM Employees WHERE EmployeeID = @TeacherID)
        BEGIN
            RAISERROR('Invalid TeacherID: no matching Employee found.', 16, 5);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        INSERT INTO OfflineVideo
            (MeetingID, VideoLink, ModuleID, VideoDuration, TeacherID)
        VALUES
            (@MeetingID, @VideoLink, @ModuleID, @VideoDuration, @TeacherID);

        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH;
END;
GO
```

6.0.69 Procedura p_CreateOrder - Emil Żychowicz

Procedura p_CreateOrder umożliwia tworzenie nowego zamówienia w systemie:

- Weryfikuje, czy @UserID istnieje w tabeli ServiceUserDetails.
- Sprawdza, czy @OrderDate nie jest datą w przyszłości.
- Automatycznie generuje unikalny OrderID.

Jeżeli dane są poprawne, nowy rekord jest dodawany do tabeli Orders. W razie błędów transakcja jest wycofywana, a użytkownik otrzymuje komunikat o błędzie.

```
CREATE OR ALTER PROCEDURE p_CreateOrder
(
    @UserID INT,
    @OrderDate DATETIME = NULL,
    @PaymentLink VARCHAR(60) = NULL
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM ServiceUserDetails WHERE ServiceUserID = @UserID)
        BEGIN
            RAISERROR('Invalid UserID: no matching user found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF @OrderDate > GETDATE()
        BEGIN
```

```
            RAISERROR('OrderDate cannot be in the future.', 16, 2);
            ROLLBACK TRANSACTION;
            RETURN;
        END;
        DECLARE @NextOrderID INT;
        IF NOT EXISTS (SELECT 1 FROM Orders)
        BEGIN
            SET @NextOrderID = 1;
        END
        ELSE
        BEGIN
            SELECT @NextOrderID = MAX(OrderID) + 1 FROM Orders;
        END;
        INSERT INTO Orders (OrderID, UserID, OrderDate, PaymentLink)
        VALUES (@NextOrderID, @UserID, @OrderDate, @PaymentLink);

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.70 Procedura p_CreateStationaryMeeting - Emil Żychowicz

Procedura p_CreateStationaryMeeting pozwala na utworzenie nowego spotkania stacjonarnego:

- Weryfikuje, czy @MeetingDate jest datą przyszłą.
- Sprawdza, czy @MeetingDuration jest większy od 00:00:00.
- Upewnia się, że @ModuleID, @RoomID i @TeacherID istnieją w odpowiednich tabelach (Modules, Rooms, Employees).
- Weryfikuje, czy @GroupSize jest większy od 0.

Jeśli dane wejściowe są poprawne, procedura dodaje nowy rekord do tabeli StationaryMeeting. W przeciwnym razie transakcja jest wycofywana.

```
CREATE OR ALTER PROCEDURE p_CreateStationaryMeeting
(
    @MeetingDate     DATETIME,
    @MeetingDuration TIME(0) = '01:30:00',
    @ModuleID        INT,
    @RoomID          INT,
    @GroupSize       INT,
    @TeacherID       INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF @MeetingDate <= GETDATE()
        BEGIN
            RAISERROR('MeetingDate must be in the future.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF @MeetingDuration <= '00:00:00'
        BEGIN
```

```
            RAISERROR('MeetingDuration must be greater than 00:00:00.', 16, 2);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF NOT EXISTS (SELECT 1 FROM Modules WHERE ModuleID = @ModuleID)
        BEGIN
            RAISERROR('Invalid ModuleID: no matching Module found.', 16, 3);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF NOT EXISTS (SELECT 1 FROM Rooms WHERE RoomID = @RoomID)
        BEGIN
            RAISERROR('Invalid RoomID: no matching Room found.', 16, 4);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF @GroupSize <= 0
        BEGIN
            RAISERROR('GroupSize must be greater than 0.', 16, 5);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF NOT EXISTS (SELECT 1 FROM Employees WHERE EmployeeID = @TeacherID)
        BEGIN
            RAISERROR('Invalid TeacherID: no matching Employee found.', 16, 6);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        DECLARE @NextMeetingID INT;
        IF NOT EXISTS (SELECT 1 FROM StationaryMeeting)
        BEGIN
            SET @NextMeetingID = 1;
        END
        ELSE
        BEGIN
            SELECT @NextMeetingID = MAX(MeetingID) + 1 FROM StationaryMeeting;
        END;

        INSERT INTO StationaryMeeting
            (MeetingID, MeetingDate, MeetingDuration, ModuleID, RoomID, GroupSize,
                TeacherID)
        VALUES
            (@NextMeetingID, @MeetingDate, @MeetingDuration, @ModuleID, @RoomID,
                @GroupSize, @TeacherID);

        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH;
END;
GO
```

6.0.71 Procedura p_DeleteClassMeetingService - Emil Żychowicz

Procedura p_DeleteClassMeetingService usuwa usługę spotkania klasowego:

- Sprawdza, czy podany @ServiceID istnieje w tabeli ClassMeetingService.
- Usuwa odpowiedni rekord, jeśli dane są poprawne.

W razie błędów transakcja jest wycofywana.

```
CREATE OR ALTER PROCEDURE p_DeleteClassMeetingService
(
    @ServiceID INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM ClassMeetingService WHERE ServiceID = @ServiceID)
        BEGIN
            RAISERROR('Invalid ServiceID: no matching service found in
                ClassMeetingService.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        DELETE FROM ClassMeetingService WHERE ServiceID = @ServiceID;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.72 Procedura p_DeleteConventionService - Emil Żychowicz

Procedura p_DeleteConventionService usuwa usługę konwencyjną:

- Weryfikuje obecność @ServiceID w tabeli ConventionService.
- Usuwa usługę, jeśli dane są prawidłowe.

W przypadku błędów użytkownik otrzymuje komunikat, a transakcja zostaje wycofana.

```
CREATE OR ALTER PROCEDURE p_DeleteConventionService
(
    @ServiceID INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM ConventionService WHERE ServiceID = @ServiceID)
        BEGIN
            RAISERROR('Invalid ServiceID: no matching service found in
                ConventionService.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        DELETE FROM ConventionService WHERE ServiceID = @ServiceID;
```

```
20
21          COMMIT TRANSACTION;
22      END TRY
23      BEGIN CATCH
24          IF @@TRANCOUNT > 0
25              ROLLBACK TRANSACTION;
26
27          THROW;
28      END CATCH;
29  END;
```

6.0.73 Procedura p_DeleteCourse - Emil Żychowicz

Procedura p_DeleteCourse usuwa kurs i wszystkie powiązane dane:

- Sprawdza, czy podany @CourseID istnieje w tabeli Courses.
- Usuwa wszystkie moduły związane z kursem, wywołując procedurę p_DeleteModule.
- Usuwa uczestników powiązanych z kursem oraz powiązania w tabeli CourseParticipants.
- Na końcu usuwa rekord kursu z tabeli Courses.

Transakcja jest wycofywana w przypadku błędów.

```
1   CREATE OR ALTER PROCEDURE p_DeleteCourse
2   (
3       @CourseID INT
4   )
5   AS
6   BEGIN
7       SET NOCOUNT ON;
8
9       BEGIN TRY
10          BEGIN TRANSACTION;
11
12          IF NOT EXISTS (SELECT 1 FROM Courses WHERE CourseID = @CourseID)
13          BEGIN
14              RAISERROR('Invalid CourseID: no matching course found.', 16, 1);
15              ROLLBACK TRANSACTION;
16              RETURN;
17          END;
18
19      DECLARE @ModuleID INT;
20      DECLARE ModulesCursor CURSOR FOR
21      SELECT ModuleID
22      FROM Modules
23      WHERE CourseID = @CourseID;
24
25      OPEN ModulesCursor;
26      FETCH NEXT FROM ModulesCursor INTO @ModuleID;
27      WHILE @@FETCH_STATUS = 0
28      BEGIN
29        EXEC p_DeleteModule @ModuleID;
30        FETCH NEXT FROM ModulesCursor INTO @ModuleID;
31      END;
32
33      CLOSE ModulesCursor;
34      DEALLOCATE ModulesCursor;
35
36
37          DELETE FROM Modules
38          WHERE CourseID = @CourseID;
39
40      DECLARE @ParticipantID INT;
41      DECLARE ParticipantsCursor CURSOR FOR
42      SELECT ParticipantID
43      FROM CourseParticipants
```

```
    WHERE CourseID = @CourseID;

    OPEN ParticipantsCursor;
    FETCH NEXT FROM ParticipantsCursor INTO @ParticipantID;

    WHILE @@FETCH_STATUS = 0
    BEGIN
      DELETE FROM CourseParticipants
      WHERE ParticipantID = @ParticipantID AND CourseID = @CourseID;
      FETCH NEXT FROM ParticipantsCursor INTO @ParticipantID;
    END;

    CLOSE ParticipantsCursor;
    DEALLOCATE ParticipantsCursor;

        DELETE FROM Courses
        WHERE CourseID = @CourseID;

        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
GO
```

6.0.74 Procedura p_DeleteCourseParticipant - Emil Żychowicz

Procedura p_DeleteCourseParticipant usuwa uczestnika z kursu:

- Sprawdza, czy @ParticipantID oraz @CourseID istnieją w tabeli CourseParticipants.
- Usuwa odpowiedni rekord, jeśli dane są poprawne.

W przeciwnym razie transakcja zostaje wycofana, a użytkownik jest informowany o błędzie.

```
CREATE OR ALTER PROCEDURE p_DeleteCourseParticipant
(
    @ParticipantID INT,
    @CourseID INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM CourseParticipants WHERE ParticipantID =
            @ParticipantID AND CourseID = @CourseID)
        BEGIN
            RAISERROR('Invalid ParticipantID or CourseID: no matching course participant
                found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        DELETE FROM CourseParticipants
        WHERE ParticipantID = @ParticipantID AND CourseID = @CourseID;

        COMMIT TRANSACTION;
    END TRY
```

```
25
26      BEGIN CATCH
27          IF @@TRANCOUNT > 0
28              ROLLBACK TRANSACTION;
29
30          THROW;
31      END CATCH;
32  END;
33  GO
```

6.0.75 Procedura p_DeleteCourseService - Emil Żychowicz

Procedura p_DeleteCourseService usuwa usługę kursową:

- Weryfikuje, czy podany @ServiceID istnieje w tabeli CourseService.
- Usuwa usługę, jeśli dane wejściowe są poprawne.

W przypadku błędów transakcja jest wycofywana, a użytkownik otrzymuje stosowny komunikat.

```
1   CREATE OR ALTER PROCEDURE p_DeleteCourseService
2   (
3       @ServiceID INT
4   )
5   AS
6   BEGIN
7       SET NOCOUNT ON;
8
9       BEGIN TRY
10          BEGIN TRANSACTION;
11
12          IF NOT EXISTS (SELECT 1 FROM CourseService WHERE ServiceID = @ServiceID)
13          BEGIN
14              RAISERROR('Invalid ServiceID: no matching service found in CourseService.',
                    16, 1);
15              ROLLBACK TRANSACTION;
16              RETURN;
17          END;
18
19          DELETE FROM CourseService WHERE ServiceID = @ServiceID;
20
21          COMMIT TRANSACTION;
22      END TRY
23      BEGIN CATCH
24          IF @@TRANCOUNT > 0
25              ROLLBACK TRANSACTION;
26
27          THROW;
28      END CATCH;
29  END;
```

6.0.76 Procedura p_DeleteModule - Emil Żychowicz

Procedura p_DeleteModule umożliwia usunięcie modułu z systemu, zapewniając spójność danych:

- Weryfikuje istnienie podanego @ModuleID w tabeli Modules.
- Usuwa powiązane dane w tabelach:
- StationaryMeeting – poprzez wywołanie p_DeleteStationaryMeeting.
- OfflineVideo – poprzez wywołanie p_DeleteOfflineVideo.
- OnlineLiveMeeting – poprzez wywołanie p_DeleteOnlineLiveMeeting.
- Usuwa rekord modułu z tabeli Modules.
- Operacja jest objęta transakcją – w przypadku błędów wszystkie zmiany są wycofywane.

```sql
CREATE OR ALTER PROCEDURE p_DeleteModule
(
    @ModuleID INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM Modules WHERE ModuleID = @ModuleID)
        BEGIN
            RAISERROR('Invalid ModuleID: no matching module found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;


    DECLARE @MeetingID INT;
    DECLARE MeetingsCursor CURSOR FOR

    SELECT MeetingID
    FROM StationaryMeeting
    WHERE ModuleID = @ModuleID

    OPEN MeetingsCursor;
    FETCH NEXT FROM MeetingsCursor INTO @MeetingID

        WHILE @@FETCH_STATUS = 0
      BEGIN
        EXEC p_DeleteStationaryMeeting @MeetingID;
        FETCH NEXT FROM MeetingsCursor INTO @MeetingID;
      END;
    CLOSE MeetingsCursor;
    DEALLOCATE MeetingsCursor;




    DECLARE @MeetingID1 INT;
    DECLARE MeetingsCursor CURSOR FOR

    SELECT MeetingID
    FROM OfflineVideo
    WHERE ModuleID = @ModuleID

    OPEN MeetingsCursor;
    FETCH NEXT FROM MeetingsCursor INTO @MeetingID1

        WHILE @@FETCH_STATUS = 0
      BEGIN
        EXEC p_DeleteOfflineVideo @MeetingID1;
        FETCH NEXT FROM MeetingsCursor INTO @MeetingID1;
      END;
    CLOSE MeetingsCursor;
    DEALLOCATE MeetingsCursor;



    DECLARE @MeetingID2 INT;
    DECLARE MeetingsCursor CURSOR FOR


    SELECT MeetingID
```

```
    FROM OnlineLiveMeeting
    WHERE ModuleID = @ModuleID

    OPEN MeetingsCursor;
    FETCH NEXT FROM MeetingsCursor INTO @MeetingID2

        WHILE @@FETCH_STATUS = 0
      BEGIN
        EXEC p_DeleteOnlineLiveMeeting @MeetingID2;
        FETCH NEXT FROM MeetingsCursor INTO @MeetingID2;
      END;
    CLOSE MeetingsCursor;
    DEALLOCATE MeetingsCursor;


        DELETE FROM Modules
        WHERE ModuleID = @ModuleID;

        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
GO
```

6.0.77 Procedura p_DeleteOfflineVideo - Emil Żychowicz

Procedura p_DeleteOfflineVideo umożliwia usunięcie nagrania offline z systemu:

- Weryfikuje istnienie podanego @MeetingID w tabeli OfflineVideo.
- Usuwa powiązane szczegóły nagrania z tabeli OfflineVideoDetails za pomocą procedury p_DeleteOfflineVideoDe
- Usuwa rekord nagrania z tabeli OfflineVideo.
- Wszystkie zmiany są objęte transakcją – w przypadku błędów są wycofywane.

```
exec delete
CREATE OR ALTER PROCEDURE p_DeleteOfflineVideo
(
    @MeetingID INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM OfflineVideo WHERE MeetingID = @MeetingID)
        BEGIN
            RAISERROR('Invalid MeetingID: no matching offline video found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        DELETE FROM OfflineVideoDetails
        WHERE MeetingID = @MeetingID;

        DELETE FROM OfflineVideo
        WHERE MeetingID = @MeetingID;

```

```
        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
GO
```

6.0.78 Procedura p_DeleteOfflineVideoDetails - Emil Żychowicz

Procedura p_DeleteOfflineVideoDetails umożliwia usunięcie szczegółów nagrania offline:

- Weryfikuje istnienie wpisu w tabeli OfflineVideoDetails na podstawie @MeetingID i @ParticipantID.
- Usuwa rekord z tabeli OfflineVideoDetails.
- Operacja jest objęta transakcją – błędy powodują wycofanie zmian.

```
CREATE OR ALTER PROCEDURE p_DeleteOfflineVideoDetails
(
    @MeetingID INT,
    @ParticipantID INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM OfflineVideoDetails WHERE MeetingID = @MeetingID
            AND ParticipantID = @ParticipantID)
        BEGIN
            RAISERROR('Invalid MeetingID or ParticipantID: no matching offline video
                details found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        DELETE FROM OfflineVideoDetails
        WHERE MeetingID = @MeetingID AND ParticipantID = @ParticipantID;

        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
GO
```

6.0.79 Procedura p_DeleteOnlineLiveMeeting - Emil Żychowicz

Procedura p_DeleteOnlineLiveMeeting służy do usunięcia spotkania online:

- Weryfikuje istnienie podanego @MeetingID w tabeli OnlineLiveMeeting.
- Usuwa powiązane szczegóły z tabeli OnlineLiveMeetingDetails za pomocą procedury p_DeleteOnlineLiveMeetin

- Usuwa rekord spotkania z tabeli OnlineLiveMeeting.
- Operacje są realizowane w transakcji – błędy powodują wycofanie zmian.

```
CREATE OR ALTER PROCEDURE p_DeleteOnlineLiveMeeting
(
    @MeetingID INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM OnlineLiveMeeting WHERE MeetingID = @MeetingID)
        BEGIN
            RAISERROR('Invalid MeetingID: no matching online live meeting found.', 16,
                1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        DELETE FROM OnlineLiveMeetingDetails
        WHERE MeetingID = @MeetingID;

        DELETE FROM OnlineLiveMeeting
        WHERE MeetingID = @MeetingID;

        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
GO
```

6.0.80 Procedura p_DeleteOnlineLiveMeetingDetails - Emil Żychowicz

Procedura p_DeleteOnlineLiveMeetingDetails umożliwia usunięcie szczegółów spotkania online:

- Sprawdza istnienie wpisu w tabeli OnlineLiveMeetingDetails na podstawie @MeetingID i @ParticipantID.
- Usuwa rekord z tabeli OnlineLiveMeetingDetails.
- Wszystkie zmiany są realizowane w transakcji – w przypadku błędów są wycofywane.

```
CREATE OR ALTER PROCEDURE p_DeleteOnlineLiveMeetingDetails
(
    @MeetingID INT,
    @ParticipantID INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM OnlineLiveMeetingDetails WHERE MeetingID =
            @MeetingID AND ParticipantID = @ParticipantID)
```

```
        BEGIN
            RAISERROR('Invalid MeetingID or ParticipantID: no matching online live
                meeting details found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        DELETE FROM OnlineLiveMeetingDetails
        WHERE MeetingID = @MeetingID AND ParticipantID = @ParticipantID;

        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
GO
```

6.0.81 Procedura p_DeleteOrder - Emil Żychowicz

Procedura p_DeleteOrder umożliwia usunięcie zamówienia z systemu:

- Sprawdza istnienie podanego @OrderID w tabeli Orders.
- Usuwa powiązane szczegóły zamówienia z tabeli OrderDetails za pomocą kursora i wywołań p_DeleteOrderDeta
- Usuwa rekord zamówienia z tabeli Orders.
- Całość jest objęta transakcją – błędy powodują wycofanie zmian.

```
CREATE OR ALTER PROCEDURE p_DeleteOrder
(
    @OrderID INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM Orders WHERE OrderID = @OrderID)
        BEGIN
            RAISERROR('Invalid OrderID: no matching order found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;


    DECLARE @ServiceID INT;
    DECLARE OrdersDetailsCursor CURSOR FOR
    SELECT ServiceID
    FROM OrderDetails
    WHERE OrderID = @OrderID;

    OPEN OrdersDetailsCursor;
    FETCH NEXT FROM OrdersDetailsCursor INTO @ServiceID;
    WHILE @@FETCH_STATUS = 0
    BEGIN
      EXEC p_DeleteOrderDetails @ServiceID, @OrderID;
      FETCH NEXT FROM OrdersDetailsCursor INTO @ServiceID;
    END;
    CLOSE OrdersDetailsCursor;
```

```
    DEALLOCATE OrdersDetailsCursor;

    DELETE FROM Orders WHERE OrderID = @OrderID;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.82 Procedura p_DeleteOrderDetails - Emil Żychowicz

Procedura p_DeleteOrderDetails służy do usunięcia szczegółów zamówienia:

- Sprawdza istnienie wpisu w tabeli OrderDetails na podstawie @OrderID i @ServiceID.
- Usuwa powiązane płatności z tabeli Payments za pomocą procedury p_DeletePayment.
- Usuwa rekord szczegółów z tabeli OrderDetails.
- Wszystkie operacje są wykonywane w transakcji – błędy powodują wycofanie zmian.

```
CREATE OR ALTER PROCEDURE p_DeleteOrderDetails
(
    @ServiceID INT,
    @OrderID INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM OrderDetails WHERE ServiceID = @ServiceID AND
            OrderID = @OrderID)
        BEGIN
            RAISERROR('Invalid ServiceID or OrderID: no matching order details found.',
                16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

    DELETE FROM Payments WHERE ServiceID = @ServiceID AND OrderID = @OrderID


        DELETE FROM OrderDetails WHERE ServiceID = @ServiceID AND OrderID = @OrderID;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.83 Procedura p_DeletePayment - Emil Żychowicz

Procedura p_DeletePayment umożliwia usunięcie płatności:

- Sprawdza istnienie podanego @PaymentID w tabeli Payments.
- Usuwa rekord płatności z tabeli Payments.
- Operacja jest objęta transakcją – w przypadku błędów zmiany są wycofywane.

```
CREATE OR ALTER PROCEDURE p_DeletePayment
(
    @PaymentID INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM Payments WHERE PaymentID = @PaymentID)
        BEGIN
            RAISERROR('Invalid PaymentID: no matching payment found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        DELETE FROM Payments WHERE PaymentID = @PaymentID;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.84 Procedura p_DeleteService - Emil Żychowicz

Procedura p_DeleteService umożliwia usunięcie usługi z systemu:

- Sprawdza istnienie @ServiceID w tabeli Services.
- Usuwa powiązane usługi zależne (CourseService, ClassMeetingService, StudiesService, WebinarService, ConventionService) za pomocą odpowiednich procedur.
- Usuwa szczegóły zamówienia powiązane z usługą z tabeli OrderDetails.
- Usuwa rekord z tabeli Services.
- Wszystkie operacje są wykonywane w transakcji – w przypadku błędów zmiany są wycofywane.

```
CREATE OR ALTER PROCEDURE p_DeleteService
(
    @ServiceID INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM Services WHERE ServiceID = @ServiceID)
        BEGIN
            RAISERROR('Invalid ServiceID: no matching service found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

```

```
    DECLARE @ServiceType VARCHAR(30);

    SELECT @ServiceType = ServiceType
    FROM Services
    WHERE ServiceID = @ServiceID;

    IF @ServiceType = 'CourseService'
    BEGIN

      EXEC p_DeleteCourseService @ServiceID;
    END
    ELSE IF @ServiceType = 'ClassMeetingService'
    BEGIN
      -- Wywołujemy procedurę p_DeleteClassMeetingService
      EXEC p_DeleteClassMeetingService @ServiceID;
    END
    ELSE IF @ServiceType = 'StudiesService'
    BEGIN
      -- Wywołujemy procedurę p_DeleteStudiesService
      EXEC p_DeleteStudiesService @ServiceID;
    END
    ELSE IF @ServiceType = 'WebinarService'
    BEGIN
      -- Wywołujemy procedurę p_DeleteWebinarService
      EXEC p_DeleteWebinarService @ServiceID;
    END
    ELSE IF @ServiceType = 'ConventionService'
    BEGIN
      -- Wywołujemy procedurę p_DeleteConventionService
      EXEC p_DeleteConventionService @ServiceID;
    END

    DECLARE @OrderID INT;
    DECLARE OrdersCursor CURSOR FOR
    SELECT OrderID
    FROM OrderDetails
    WHERE ServiceID = @ServiceID;

    OPEN OrdersCursor;
    FETCH NEXT FROM OrdersCursor INTO @OrderID;
    WHILE @@FETCH_STATUS = 0
    BEGIN
      EXEC p_DeleteOrderDetails @ServiceID, @OrderID
      FETCH NEXT FROM OrdersCursor INTO @OrderID;
    END;
    CLOSE OrdersCursor;
    DEALLOCATE OrdersCursor;
    -- Usuwamy rekord z Services
    DELETE FROM Services WHERE ServiceID = @ServiceID;

    COMMIT TRANSACTION;
  END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.85 Procedura p_DeleteStationaryMeeting - Emil Żychowicz

Procedura p_DeleteStationaryMeeting umożliwia usunięcie spotkania stacjonarnego wraz z jego szczegółami:

- Sprawdza, czy podany @MeetingID istnieje w tabeli StationaryMeeting.
- W przypadku nieistniejącego @MeetingID wyświetlany jest komunikat o błędzie, a transakcja jest wycofywana.
- Usuwa szczegóły spotkania z tabeli StationaryMeetingDetails.
- Usuwa rekord spotkania z tabeli StationaryMeeting.
- Transakcja jest wycofywana w przypadku wystąpienia błędu.

```
CREATE OR ALTER PROCEDURE p_DeleteStationaryMeeting
(
    @MeetingID INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        -- Validate MeetingID
        IF NOT EXISTS (SELECT 1 FROM StationaryMeeting WHERE MeetingID = @MeetingID)
        BEGIN
            RAISERROR('Invalid MeetingID: no matching stationary meeting found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        DELETE FROM StationaryMeetingDetails
        WHERE MeetingID = @MeetingID;

        -- Delete the StationaryMeeting
        DELETE FROM StationaryMeeting
        WHERE MeetingID = @MeetingID;


        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
GO
```

6.0.86 Procedura p_DeleteStationaryMeetingDetails - Emil Żychowicz

Procedura p_DeleteStationaryMeetingDetails umożliwia usunięcie szczegółowych danych dotyczących uczestnictwa w spotkaniu:

- Weryfikuje, czy kombinacja @MeetingID i @ParticipantID istnieje w tabeli StationaryMeetingDetails.
- Wyświetla komunikat o błędzie w przypadku braku dopasowania danych, wycofując transakcję.
- Usuwa szczegóły z tabeli StationaryMeetingDetails. Wycofuje transakcję w razie błędu.

```
CREATE OR ALTER PROCEDURE p_DeleteStationaryMeetingDetails
(
    @MeetingID INT,
    @ParticipantID INT
)
AS
BEGIN
    SET NOCOUNT ON;
```

```
9
10      BEGIN TRY
11          BEGIN TRANSACTION;
12
13          -- Validate MeetingID and ParticipantID
14          IF NOT EXISTS (SELECT 1 FROM StationaryMeetingDetails WHERE MeetingID =
                @MeetingID AND ParticipantID = @ParticipantID)
15          BEGIN
16              RAISERROR('Invalid MeetingID or ParticipantID: no matching details found.',
                    16, 1);
17              ROLLBACK TRANSACTION;
18              RETURN;
19          END;
20
21          -- Delete the StationaryMeetingDetails entry
22          DELETE FROM StationaryMeetingDetails
23          WHERE MeetingID = @MeetingID AND ParticipantID = @ParticipantID;
24
25          COMMIT TRANSACTION;
26      END TRY
27
28      BEGIN CATCH
29          IF @@TRANCOUNT > 0
30              ROLLBACK TRANSACTION;
31
32          THROW;
33      END CATCH;
34  END;
35  GO
```

6.0.87 Procedura p_DeleteStudiesService - Emil Żychowicz

Procedura p_DeleteStudiesService służy do usuwania rekordów z tabeli StudiesService:

- Weryfikuje, czy podany @ServiceID istnieje w tabeli StudiesService. W przypadku braku rekordu wyświetlany jest błąd, a transakcja jest wycofywana.
- Usuwa odpowiedni rekord z tabeli StudiesService.
- Obsługuje wyjątki, wycofując transakcję w przypadku wystąpienia błędu.

```
1   CREATE OR ALTER PROCEDURE p_DeleteStudiesService
2   (
3       @ServiceID INT
4   )
5   AS
6   BEGIN
7       SET NOCOUNT ON;
8
9       BEGIN TRY
10          BEGIN TRANSACTION;
11
12          -- Sprawdzamy, czy ServiceID istnieje w tabeli StudiesService
13          IF NOT EXISTS (SELECT 1 FROM StudiesService WHERE ServiceID = @ServiceID)
14          BEGIN
15              RAISERROR('Invalid ServiceID: no matching service found in StudiesService.',
                    16, 1);
16              ROLLBACK TRANSACTION;
17              RETURN;
18          END;
19
20          -- Usuwamy rekord z StudiesService
21          DELETE FROM StudiesService WHERE ServiceID = @ServiceID;
22
23          COMMIT TRANSACTION;
24      END TRY
```

```
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.88 Procedura p_DeleteWebinarService - Emil Żychowicz

Procedura p_DeleteWebinarService umożliwia usunięcie rekordu z tabeli WebinarService:

- Sprawdza obecność @ServiceID w tabeli WebinarService.
- Jeśli @ServiceID nie istnieje, wyświetlany jest komunikat o błędzie, a transakcja jest wycofywana.
- Usuwa rekord z tabeli WebinarService.
- Wycofuje transakcję w razie wystąpienia błędu.

```
CREATE OR ALTER PROCEDURE p_DeleteWebinarService
(
    @ServiceID INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        -- Sprawdzamy, czy ServiceID istnieje w tabeli WebinarService
        IF NOT EXISTS (SELECT 1 FROM WebinarService WHERE ServiceID = @ServiceID)
        BEGIN
            RAISERROR('Invalid ServiceID: no matching service found in WebinarService.',
                16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        -- Usuwamy rekord z WebinarService
        DELETE FROM WebinarService WHERE ServiceID = @ServiceID;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.89 Procedura p_EditClassMeetingService - Emil Żychowicz

Procedura p_EditClassMeetingService umożliwia edycję danych w tabeli ClassMeetingService:

- Weryfikuje, czy @ServiceID istnieje w tabeli.
- Sprawdza, czy podane ceny (@PriceStudents, @PriceOthers) są większe od 0.
- Aktualizuje tylko te pola, dla których podano nowe wartości. W przypadku błędu transakcja jest wycofywana.

```
CREATE OR ALTER PROCEDURE p_EditClassMeetingService
(
    @ServiceID INT,
```

```
    @PriceStudents MONEY = NULL,   -- Domyślnie NULL
    @PriceOthers MONEY = NULL      -- Domyślnie NULL
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        -- Sprawdzamy, czy ServiceID istnieje w tabeli ClassMeetingService
        IF NOT EXISTS (SELECT 1 FROM ClassMeetingService WHERE ServiceID = @ServiceID)
        BEGIN
            RAISERROR('Invalid ServiceID: no matching service found in
                ClassMeetingService.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        -- Sprawdzamy, czy ceny są większe od 0, zgodnie z CHECK constraint
        IF @PriceStudents IS NOT NULL AND @PriceStudents <= 0
        BEGIN
            RAISERROR('PriceStudents must be greater than 0.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF @PriceOthers IS NOT NULL AND @PriceOthers <= 0
        BEGIN
            RAISERROR('PriceOthers must be greater than 0.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        -- Edytujemy rekord w tabeli ClassMeetingService, aktualizujemy tylko, gdy
            wartości są przekazane
        UPDATE ClassMeetingService
        SET
            PriceStudents = CASE WHEN @PriceStudents IS NOT NULL THEN @PriceStudents
                ELSE PriceStudents END,
            PriceOthers = CASE WHEN @PriceOthers IS NOT NULL THEN @PriceOthers ELSE
                PriceOthers END
        WHERE ServiceID = @ServiceID;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.90 Procedura p_EditConventionService - Emil Żychowicz

Procedura p_EditConventionService umożliwia edycję ceny w tabeli ConventionService:

- Sprawdza, czy @ServiceID istnieje w tabeli.
- Weryfikuje, czy wartość @Price jest większa od 0.
- Aktualizuje wartość Price w tabeli ConventionService.
- Obsługuje wyjątki, wycofując transakcję w przypadku błędu.

```sql
CREATE OR ALTER PROCEDURE p_EditConventionService
(
    @ServiceID INT,
    @Price MONEY
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        -- Sprawdzamy, czy ServiceID istnieje w tabeli ConventionService
        IF NOT EXISTS (SELECT 1 FROM ConventionService WHERE ServiceID = @ServiceID)
        BEGIN
            RAISERROR('Invalid ServiceID: no matching service found in
                ConventionService.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        -- Sprawdzamy, czy Price jest większe od 0
        IF @Price <= 0
        BEGIN
            RAISERROR('Price must be greater than 0.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        -- Edytujemy rekord w tabeli ConventionService
        UPDATE ConventionService
        SET Price = @Price
        WHERE ServiceID = @ServiceID;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.91 Procedura p_EditCourses - Emil Żychowicz

Procedura p_EditCourses umożliwia edycję informacji o kursach w tabeli Courses:

- Weryfikuje, czy @CourseID istnieje w tabeli.
- Aktualizuje następujące pola (o ile przekazano odpowiednie wartości): nazwa kursu, opis, koordynator, data rozpoczęcia, limit zapisów.
- Waliduje wartości, takie jak długość tekstów, zakres dat czy minimalny limit zapisów.
- Wycofuje transakcję w razie nieprawidłowych danych lub błędu.

```sql
CREATE OR ALTER PROCEDURE p_EditCourses
(
    @CourseID INT,
    @CourseName VARCHAR(40) = NULL,
    @CourseDescription VARCHAR(255) = NULL,
    @CourseCoordinatorID INT = NULL,
    @CourseDate DATE = NULL,
    @EnrollmentLimit INT = NULL
)
```

```
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        -- Sprawdzenie, czy kurs istnieje
        IF NOT EXISTS (SELECT 1 FROM Courses WHERE CourseID = @CourseID)
        BEGIN
            RAISERROR('Invalid CourseID: no matching course found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        -- Walidacja i aktualizacja CourseName
        IF @CourseName IS NOT NULL
        BEGIN
            IF @CourseName = ''
            BEGIN
                RAISERROR('CourseName cannot be empty.', 16, 2);
                ROLLBACK TRANSACTION;
                RETURN;
            END;
            UPDATE Courses
            SET CourseName = @CourseName
            WHERE CourseID = @CourseID;
        END;


        IF @CourseDescription IS NOT NULL
        BEGIN
            UPDATE Courses
            SET CourseDescription = @CourseDescription
            WHERE CourseID = @CourseID;
        END;


        IF @CourseCoordinatorID IS NOT NULL
        BEGIN
            IF @CourseCoordinatorID <= 0
            BEGIN
                RAISERROR('Invalid CourseCoordinatorID: must be greater than 0.', 16, 3);
                ROLLBACK TRANSACTION;
                RETURN;
            END;
            UPDATE Courses
            SET CourseCoordinatorID = @CourseCoordinatorID
            WHERE CourseID = @CourseID;
        END;


        IF @CourseDate IS NOT NULL
        BEGIN
            IF @CourseDate <= '2015-01-01' OR @CourseDate >= '2030-01-01'
            BEGIN
                RAISERROR('CourseDate must be between 01-01-2015 and 01-01-2030.', 16,
                    5);
                ROLLBACK TRANSACTION;
                RETURN;
            END;
            UPDATE Courses
            SET CourseDate = @CourseDate
            WHERE CourseID = @CourseID;
        END;
```

```
74
75
76          IF @EnrollmentLimit IS NOT NULL
77          BEGIN
78              IF @EnrollmentLimit <= 1
79              BEGIN
80                  RAISERROR('EnrollmentLimit must be greater than 1.', 16, 6);
81                  ROLLBACK TRANSACTION;
82                  RETURN;
83              END;
84              UPDATE Courses
85              SET EnrollmentLimit = @EnrollmentLimit
86              WHERE CourseID = @CourseID;
87          END;
88
89
90          COMMIT TRANSACTION;
91      END TRY
92      BEGIN CATCH
93          IF @@TRANCOUNT > 0
94              ROLLBACK TRANSACTION;
95
96          THROW;
97      END CATCH;
98  END;
```

6.0.92 Procedura p_EditCourseService - Emil Żychowicz

Procedura p_EditCourseService umożliwia edycję danych w tabeli CourseService:

- Weryfikuje, czy @ServiceID istnieje w tabeli.
- Sprawdza poprawność wartości @AdvanceValue (większa od 0) i @FullPrice (większa lub równa @AdvanceValue).
- Aktualizuje pola tylko w przypadku przekazania nowych wartości.
- Wycofuje transakcję w przypadku błędu.

```
1   CREATE OR ALTER PROCEDURE p_EditCourseService
2   (
3       @ServiceID INT,
4       @AdvanceValue MONEY = NULL,   -- Domyślnie NULL
5       @FullPrice MONEY = NULL       -- Domyślnie NULL
6   )
7   AS
8   BEGIN
9       SET NOCOUNT ON;
10
11      BEGIN TRY
12          BEGIN TRANSACTION;
13
14          -- Sprawdzamy, czy ServiceID istnieje w tabeli CourseService
15          IF NOT EXISTS (SELECT 1 FROM CourseService WHERE ServiceID = @ServiceID)
16          BEGIN
17              RAISERROR('Invalid ServiceID: no matching service found in CourseService.',
                    16, 1);
18              ROLLBACK TRANSACTION;
19              RETURN;
20          END;
21
22          -- Sprawdzamy, czy AdvanceValue i FullPrice są większe od 0 oraz czy FullPrice
                >= AdvanceValue
23          IF @AdvanceValue IS NOT NULL AND @AdvanceValue <= 0
24          BEGIN
25              RAISERROR('AdvanceValue must be greater than 0.', 16, 1);
26              ROLLBACK TRANSACTION;
```

```
            RETURN;
        END;

        IF @FullPrice IS NOT NULL AND @FullPrice < @AdvanceValue
        BEGIN
            RAISERROR('FullPrice must be greater than or equal to AdvanceValue.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        -- Edytujemy rekord w tabeli CourseService, aktualizujemy tylko, gdy wartości są przekazane
        UPDATE CourseService
        SET
            AdvanceValue = CASE WHEN @AdvanceValue IS NOT NULL THEN @AdvanceValue ELSE AdvanceValue END,
            FullPrice = CASE WHEN @FullPrice IS NOT NULL THEN @FullPrice ELSE FullPrice END
        WHERE ServiceID = @ServiceID;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.93 Procedura p_EditModules - Emil Żychowicz

Procedura p_EditModules umożliwia edycję informacji o modułach w tabeli Modules:

- Sprawdza, czy @ModuleID istnieje w tabeli.
- Aktualizuje pola, takie jak: język, kurs, tłumacz, koordynator modułu, typ modułu (jeśli przekazano wartości).
- Wyświetla błąd w przypadku pustego @ModuleType.
- Wycofuje transakcję w razie błędu.

```
CREATE OR ALTER PROCEDURE p_EditModules
(
    @ModuleID INT,
    @LanguageID INT = NULL,
    @CourseID INT = NULL,
    @TranslatorID INT = NULL,
    @ModuleCoordinatorID INT = NULL,
    @ModuleType VARCHAR(30) = NULL
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM Modules WHERE ModuleID = @ModuleID)
        BEGIN
            RAISERROR('Invalid ModuleID: no matching module found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF @LanguageID IS NOT NULL
```

```
        BEGIN
            UPDATE Modules
            SET LanguageID = @LanguageID
            WHERE ModuleID = @ModuleID;
        END;

        IF @CourseID IS NOT NULL
        BEGIN
            UPDATE Modules
            SET CourseID = @CourseID
            WHERE ModuleID = @ModuleID;
        END;

        IF @TranslatorID IS NOT NULL
        BEGIN
            UPDATE Modules
            SET TranslatorID = @TranslatorID
            WHERE ModuleID = @ModuleID;
        END;

        IF @ModuleCoordinatorID IS NOT NULL
        BEGIN
            UPDATE Modules
            SET ModuleCoordinatorID = @ModuleCoordinatorID
            WHERE ModuleID = @ModuleID;
        END;

        IF @ModuleType IS NOT NULL
        BEGIN
            IF @ModuleType = ''
            BEGIN
                RAISERROR('ModuleType cannot be empty.', 16, 2);
                ROLLBACK TRANSACTION;
                RETURN;
            END;
            UPDATE Modules
            SET ModuleType = @ModuleType
            WHERE ModuleID = @ModuleID;
        END;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.94 Procedura p_EditOfflineVideo - Emil Żychowicz

Procedura p_EditOfflineVideo umożliwia edycję informacji o wideo offline w tabeli OfflineVideo:

- Weryfikuje, czy @MeetingID istnieje w tabeli.
- Aktualizuje pola, takie jak: link do wideo, czas trwania, identyfikator nauczyciela (jeśli przekazano wartości).
- Wycofuje transakcję w przypadku błędu.

```
CREATE OR ALTER PROCEDURE p_EditOfflineVideo
(
    @MeetingID INT,
    @VideoLink VARCHAR(60) = NULL,
    @VideoDuration TIME(0) = NULL,
```

```
    @TeacherID INT = NULL
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM OfflineVideo WHERE MeetingID = @MeetingID)
        BEGIN
            RAISERROR('Invalid MeetingID: no matching offline video found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF @VideoLink IS NOT NULL
        BEGIN
            UPDATE OfflineVideo
            SET VideoLink = @VideoLink
            WHERE MeetingID = @MeetingID;
        END;

        IF @VideoDuration IS NOT NULL
        BEGIN
            UPDATE OfflineVideo
            SET VideoDuration = @VideoDuration
            WHERE MeetingID = @MeetingID;
        END;

        IF @TeacherID IS NOT NULL
        BEGIN
            UPDATE OfflineVideo
            SET TeacherID = @TeacherID
            WHERE MeetingID = @MeetingID;
        END;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.95 Procedura p_EditOfflineVideoDateOfViewing - Emil Żychowicz

Procedura p_EditOfflineVideoDateOfViewing służy do aktualizacji daty obejrzenia wideo offline:

- Sprawdza, czy rekord z @MeetingID i @ParticipantID istnieje w tabeli OfflineVideoDetails.
- Aktualizuje pole DateOfViewing.
- Wycofuje transakcję w razie błędu.

```
CREATE OR ALTER PROCEDURE p_EditOfflineVideoDateOfViewing
(
    @MeetingID INT,
    @ParticipantID INT,
    @DateOfViewing DATE
)
AS
BEGIN
    SET NOCOUNT ON;
```

```
10
11      BEGIN TRY
12          BEGIN TRANSACTION;
13
14          IF NOT EXISTS (SELECT 1 FROM OfflineVideoDetails WHERE MeetingID = @MeetingID
                AND ParticipantID = @ParticipantID)
15          BEGIN
16              RAISERROR('Invalid MeetingID or ParticipantID: no matching record found.',
                    16, 1);
17              ROLLBACK TRANSACTION;
18              RETURN;
19          END;
20
21
22          UPDATE OfflineVideoDetails
23          SET DateOfViewing = @DateOfViewing
24          WHERE MeetingID = @MeetingID AND ParticipantID = @ParticipantID;
25
26
27          COMMIT TRANSACTION;
28      END TRY
29      BEGIN CATCH
30          IF @@TRANCOUNT > 0
31              ROLLBACK TRANSACTION;
32
33          THROW;
34      END CATCH;
35  END;
```

6.0.96 Procedura p_EditOnlineLiveAttendance - Emil Żychowicz

Procedura p_EditOnlineLiveAttendance umożliwia edycję informacji o obecności uczestnika na spotkaniu online:

- Weryfikuje, czy kombinacja @MeetingID i @ParticipantID istnieje w tabeli OnlineLiveMeetingDetails.
- Aktualizuje pole Attendance.
- Obsługuje wyjątki i wycofuje transakcję w przypadku błędu.

```
1   CREATE OR ALTER PROCEDURE p_EditOnlineLiveAttendance
2   (
3       @MeetingID INT,
4       @ParticipantID INT,
5       @Attendance BIT
6   )
7   AS
8   BEGIN
9       SET NOCOUNT ON;
10
11      BEGIN TRY
12          BEGIN TRANSACTION;
13
14          IF NOT EXISTS (SELECT 1 FROM OnlineLiveMeetingDetails WHERE MeetingID =
                @MeetingID AND ParticipantID = @ParticipantID)
15          BEGIN
16              RAISERROR('Invalid MeetingID or ParticipantID: no matching record found.',
                    16, 1);
17              ROLLBACK TRANSACTION;
18              RETURN;
19          END;
20
21
22          UPDATE OnlineLiveMeetingDetails
23          SET Attendance = @Attendance
24          WHERE MeetingID = @MeetingID AND ParticipantID = @ParticipantID;
```

```
25
26
27          COMMIT TRANSACTION;
28      END TRY
29      BEGIN CATCH
30          IF @@TRANCOUNT > 0
31              ROLLBACK TRANSACTION;
32          THROW;
33      END CATCH;
34  END;
```

6.0.97 Procedura p_EditOnlineLiveMeeting - Emil Żychowicz

Procedura p_EditOnlineLiveMeeting umożliwia edycję danych o spotkaniach online na żywo:

- Sprawdza obecność @MeetingID w tabeli OnlineLiveMeeting.
- Aktualizuje następujące pola (jeśli podano wartości): nazwa platformy, link, link do nagrania, data spotkania, czas trwania, identyfikator nauczyciela.
- Wycofuje transakcję w przypadku błędu.

```
1   CREATE OR ALTER PROCEDURE p_EditOnlineLiveMeeting
2   (
3       @MeetingID INT,
4       @PlatformName VARCHAR(20) = NULL,
5       @Link VARCHAR(60) = NULL,
6       @VideoLink VARCHAR(60) = NULL,
7       @MeetingDate DATETIME = NULL,
8       @MeetingDuration TIME(0) = NULL,
9       @TeacherID INT = NULL
10  )
11  AS
12  BEGIN
13      SET NOCOUNT ON;
14
15      BEGIN TRY
16          BEGIN TRANSACTION;
17
18          IF NOT EXISTS (SELECT 1 FROM OnlineLiveMeeting WHERE MeetingID = @MeetingID)
19          BEGIN
20              RAISERROR('Invalid MeetingID: no matching online live meeting found.', 16,
                    1);
21              ROLLBACK TRANSACTION;
22              RETURN;
23          END;
24
25          IF @PlatformName IS NOT NULL
26          BEGIN
27              UPDATE OnlineLiveMeeting
28              SET PlatformName = @PlatformName
29              WHERE MeetingID = @MeetingID;
30          END;
31
32          IF @Link IS NOT NULL
33          BEGIN
34              UPDATE OnlineLiveMeeting
35              SET Link = @Link
36              WHERE MeetingID = @MeetingID;
37          END;
38
39          IF @VideoLink IS NOT NULL
40          BEGIN
41              UPDATE OnlineLiveMeeting
42              SET VideoLink = @VideoLink
43              WHERE MeetingID = @MeetingID;
```

```
        END;

        IF @MeetingDate IS NOT NULL
        BEGIN
            UPDATE OnlineLiveMeeting
            SET MeetingDate = @MeetingDate
            WHERE MeetingID = @MeetingID;
        END;

        IF @MeetingDuration IS NOT NULL
        BEGIN
            UPDATE OnlineLiveMeeting
            SET MeetingDuration = @MeetingDuration
            WHERE MeetingID = @MeetingID;
        END;

        IF @TeacherID IS NOT NULL
        BEGIN
            UPDATE OnlineLiveMeeting
            SET TeacherID = @TeacherID
            WHERE MeetingID = @MeetingID;
        END;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.98 Procedura p_EditPayment - Emil Żychowicz

Procedura ta pozwala na edytowanie informacji o płatności. Przyjmuje następujące parametry:

- @PaymentID: identyfikator płatności, która ma zostać edytowana.
- @PaymentValue: nowa wartość płatności (opcjonalna).
- @PaymentDate: nowa data płatności (opcjonalna).

Działania:

- Sprawdza, czy płatność o podanym PaymentID istnieje w tabeli Payments.
- Weryfikuje, czy wartość płatności jest większa niż 0, jeśli została przekazana.
- Aktualizuje odpowiednie kolumny w tabeli Payments, jeśli nowe wartości zostały przekazane.
- Wszystkie operacje są realizowane w ramach transakcji, a w przypadku błędu następuje jej rollback.

```
CREATE OR ALTER PROCEDURE p_EditPayment
(
    @PaymentID INT,
    @PaymentValue MONEY = NULL,  -- Domyślnie NULL
    @PaymentDate DATETIME = NULL -- Domyślnie NULL
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        -- Sprawdzamy, czy PaymentID istnieje w tabeli Payments
        IF NOT EXISTS (SELECT 1 FROM Payments WHERE PaymentID = @PaymentID)
        BEGIN
```

```
            RAISERROR('Invalid PaymentID: no matching payment found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        -- Sprawdzamy, czy PaymentValue jest większe od 0, jeśli nie jest NULL
        IF @PaymentValue IS NOT NULL AND @PaymentValue <= 0
        BEGIN
            RAISERROR('PaymentValue must be greater than 0.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        -- Edytujemy rekord w tabeli Payments, aktualizujemy tylko, gdy wartości są przekazane
        UPDATE Payments
        SET
            PaymentValue = CASE WHEN @PaymentValue IS NOT NULL THEN @PaymentValue ELSE PaymentValue END,
            PaymentDate = CASE WHEN @PaymentDate IS NOT NULL THEN @PaymentDate ELSE PaymentDate END
        WHERE PaymentID = @PaymentID;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.99 Procedura p_EditStationaryMeeting - Emil Żychowicz

Procedura ta służy do edytowania informacji o spotkaniu stacjonarnym. Parametry procedury to:

- @MeetingID: identyfikator spotkania.
- @MeetingDate: nowa data spotkania (opcjonalna).
- @MeetingDuration: czas trwania spotkania (opcjonalny).
- @RoomID: identyfikator sali (opcjonalny).
- @GroupSize: wielkość grupy (opcjonalna).
- @TeacherID: identyfikator nauczyciela (opcjonalny).

Działania:

- Sprawdza, czy spotkanie o podanym MeetingID istnieje w tabeli StationaryMeeting.
- Jeśli wartości takie jak data spotkania, czas trwania, sala, wielkość grupy lub nauczyciel zostały przekazane, procedura je aktualizuje.
- Weryfikuje, czy liczba uczestników nie przekracza pojemności przypisanej sali.
- Wszystkie operacje są przeprowadzane w transakcji, która jest wycofywana w razie błędu.

```
CREATE OR ALTER PROCEDURE p_EditStationaryMeeting
(
    @MeetingID INT,
    @MeetingDate DATETIME = NULL,
    @MeetingDuration TIME(0) = NULL,
    @RoomID INT = NULL,
    @GroupSize INT = NULL,
    @TeacherID INT = NULL
)
AS
BEGIN
```

```
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM StationaryMeeting WHERE MeetingID = @MeetingID)
        BEGIN
            RAISERROR('Invalid MeetingID: no matching meeting found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF @MeetingDate IS NOT NULL
        BEGIN
            UPDATE StationaryMeeting
            SET MeetingDate = @MeetingDate
            WHERE MeetingID = @MeetingID;
        END;

        IF @MeetingDuration IS NOT NULL
        BEGIN
            UPDATE StationaryMeeting
            SET MeetingDuration = @MeetingDuration
            WHERE MeetingID = @MeetingID;
        END;


    IF @GroupSize <= (SELECT Capacity FROM Rooms WHERE RoomID = @RoomID)
        BEGIN
            RAISERROR('Incompatible Room: not enough capacity', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF @RoomID IS NOT NULL
        BEGIN
            UPDATE StationaryMeeting
            SET RoomID = @RoomID
            WHERE MeetingID = @MeetingID;
        END;

        IF @GroupSize IS NOT NULL
        BEGIN
            UPDATE StationaryMeeting
            SET GroupSize = @GroupSize
            WHERE MeetingID = @MeetingID;
        END;

        IF @TeacherID IS NOT NULL
        BEGIN
            UPDATE StationaryMeeting
            SET TeacherID = @TeacherID
            WHERE MeetingID = @MeetingID;
        END;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.100 Procedura p_EditStationaryMeetingAttendance - Emil Żychowicz

Procedura ta umożliwia edytowanie obecności uczestników na spotkaniach stacjonarnych. Parametry procedury to:

- @MeetingID: identyfikator spotkania.
- @ParticipantID: identyfikator uczestnika.
- @Attendance: wartość obecności (TRUE lub FALSE).

Działania:

- Sprawdza, czy rekord uczestnika dla danego spotkania istnieje w tabeli StationaryMeetingDetails.
- Jeśli parametr @Attendance jest przekazany, procedura aktualizuje obecność danego uczestnika. Operacja odbywa się w ramach transakcji, a w przypadku błędu następuje rollback.

```
CREATE OR ALTER PROCEDURE p_EditStationaryMeetingAttendance
(
    @MeetingID INT,
    @ParticipantID INT,
    @Attendance BIT = NULL
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM StationaryMeetingDetails WHERE MeetingID =
            @MeetingID AND ParticipantID = @ParticipantID)
        BEGIN
            RAISERROR('Invalid MeetingID or ParticipantID: no matching record found.',
                16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        IF @Attendance IS NOT NULL
        BEGIN
            UPDATE StationaryMeetingDetails
            SET Attendance = @Attendance
            WHERE MeetingID = @MeetingID AND ParticipantID = @ParticipantID;
        END;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.101 Procedura p_EditStudiesService - Emil Żychowicz

Procedura ta umożliwia edytowanie opłat za usługi związane z nauką. Parametry procedury to:

- @ServiceID: identyfikator usługi.
- @EntryFee: nowa opłata za usługę.

Działania:

- Sprawdza, czy ServiceID istnieje w tabeli StudiesService.

- Weryfikuje, czy EntryFee jest większe od 0.
- Aktualizuje wartość opłaty za usługę w tabeli StudiesService.
- Cała operacja realizowana jest w ramach transakcji, której wycofanie jest uruchamiane w przypadku napotkania błędu.

```sql
CREATE OR ALTER PROCEDURE p_EditStudiesService
(
    @ServiceID INT,
    @EntryFee MONEY
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        -- Sprawdzamy, czy ServiceID istnieje w tabeli StudiesService
        IF NOT EXISTS (SELECT 1 FROM StudiesService WHERE ServiceID = @ServiceID)
        BEGIN
            RAISERROR('Invalid ServiceID: no matching service found in StudiesService.',
                16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        -- Sprawdzamy, czy EntryFee jest większe od 0
        IF @EntryFee <= 0
        BEGIN
            RAISERROR('EntryFee must be greater than 0.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        -- Edytujemy rekord w tabeli StudiesService
        UPDATE StudiesService
        SET EntryFee = @EntryFee
        WHERE ServiceID = @ServiceID;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.102 Procedura p_EditWebinarService - Emil Żychowicz

Procedura ta pozwala na edytowanie cen usług webinarowych. Parametry:

- @ServiceID: identyfikator usługi.
- @Price: nowa cena usługi.

Działania:

- Sprawdza, czy ServiceID istnieje w tabeli WebinarService.
- Weryfikuje, czy Price jest większe od 0.
- Aktualizuje wartość ceny za usługę w tabeli WebinarService.
- Operacja jest przeprowadzana w ramach transakcji, która jest wycofywana w przypadku błędu.

```sql
CREATE OR ALTER PROCEDURE p_EditWebinarService
(
    @ServiceID INT,
    @Price MONEY
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        -- Sprawdzamy, czy ServiceID istnieje w tabeli WebinarService
        IF NOT EXISTS (SELECT 1 FROM WebinarService WHERE ServiceID = @ServiceID)
        BEGIN
            RAISERROR('Invalid ServiceID: no matching service found in WebinarService.',
                16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        -- Sprawdzamy, czy Price jest większe od 0
        IF @Price <= 0
        BEGIN
            RAISERROR('Price must be greater than 0.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        -- Edytujemy rekord w tabeli WebinarService
        UPDATE WebinarService
        SET Price = @Price
        WHERE ServiceID = @ServiceID;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.103 Procedura p_FinalizeOrder - Emil Żychowicz

Procedura ta finalizuje zamówienie, przypisując link do płatności. Parametry:

- @OrderID: identyfikator zamówienia.
- @PaymentLink: link do płatności.

Działania:

- Sprawdza, czy zamówienie o podanym OrderID istnieje.
- Jeśli link do płatności jest przekazany, aktualizuje odpowiednią kolumnę w tabeli Orders.
- Zawiera skomentowany kod umożliwiający dodanie płatności dla usług związanych z zamówieniem.
- Cała operacja realizowana jest w ramach transakcji, która zostanie wycofana w przypadku napotkania błędu.

```sql
CREATE OR ALTER PROCEDURE p_FinalizeOrder
(
    @OrderID INT,
    @PaymentLink VARCHAR(60)
```

```
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM Orders WHERE OrderID = @OrderID)
        BEGIN
            RAISERROR('Invalid OrderID: no matching order found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        UPDATE Orders
        SET
            PaymentLink = CASE WHEN @PaymentLink IS NOT NULL THEN @PaymentLink ELSE
                PaymentLink END,
            OrderDate = GETDATE()
        WHERE OrderID = @OrderID;

    --DECLARE OrderService CURSOR FOR
    --SELECT ServiceID
    --FROM OrderDetails
    --WHERE OrderID = @OrderID;

    --DECLARE @ServiceID INT;
    --OPEN OrderService;
    --FETCH NEXT FROM OrderService INTO @ServiceID;
    --WHILE @@FETCH_STATUS = 0
    --BEGIN
    --   exec p_AddPayment 0, NULL, @ServiceID, @OrderID;
    --   FETCH NEXT FROM OrderService INTO @ServiceID
    --END;

    --CLOSE OrderService;
    --DEALLOCATE OrderService;


        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.104 Procedura p_UpdatePrincipalAgreement - Emil Żychowicz

Procedura ta umożliwia aktualizację zgody głównej na usługi w ramach zamówienia. Parametry:

- @OrderID: identyfikator zamówienia.
- @ServiceID: identyfikator usługi.
- @PrincipalAgreement: nowa wartość zgody głównej (TRUE lub FALSE).

Działania:

- Sprawdza, czy rekord o podanym OrderID i ServiceID istnieje w tabeli OrderDetails.
- Zmienia wartość PrincipalAgreement w tabeli OrderDetails na nową.
- Cała operacja odbywa się w ramach transakcji, a w przypadku błędu następuje jej rollback.

```sql
CREATE OR ALTER PROCEDURE p_UpdatePrincipalAgreement
(
    @OrderID INT,
    @ServiceID INT,
    @PrincipalAgreement BIT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        -- Sprawdzenie, czy rekord istnieje
        IF NOT EXISTS (SELECT 1 FROM OrderDetails WHERE OrderID = @OrderID AND ServiceID
            = @ServiceID)
        BEGIN
            RAISERROR('No such a record in OrderDetails', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        -- Aktualizacja wartości PrincipalAgreement
        UPDATE OrderDetails
        SET PrincipalAgreement = @PrincipalAgreement
        WHERE OrderID = @OrderID AND ServiceID = @ServiceID;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH;
END;
GO
```

6.0.105 Procedura p_AddWebinarUser - Emil Żychowicz

Procedura ta umożliwia zapisanie użytkownika na webinar. Parametry:

- @UserID: identyfikator użytkownika.
- @WebinarID: identyfikator webinaru.

Działania:

- Sprawdza, czy webinar o podanym WebinarID istnieje.
- Jeśli użytkownik jest już zapisany na dany webinar, operacja zostaje przerwana.
- Jeśli użytkownik nie jest zapisany, dodaje go do tabeli WebinarDetails i ustawia datę ważności dostępu do webinaru na 30 dni.
- Operacja jest przeprowadzana w ramach transakcji, a w przypadku błędu następuje rollback.

```sql
CREATE PROCEDURE p_AddWebinarUser
    @UserID int,
    @WebinarID int
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION;
        IF NOT EXISTS (
            SELECT 1
            FROM Webinars
            WHERE WebinarID = @WebinarID
```

```
        )
        BEGIN
            RAISERROR('No such a webinar in Webinars', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END

        IF EXISTS (
            SELECT 1
            FROM WebinarDetails
            WHERE UserID = @UserID AND WebinarID = @WebinarID
        )
        BEGIN
            RAISERROR('User already enrolled.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END
    DECLARE @AvailableUntil DATETIME;
    SET @AvailableUntil = DATEADD(DAY, 30, GETDATE()) -- dostepne na 30dni
        INSERT INTO WebinarDetails(UserID, WebinarID,AvailableDue)
        VALUES (@UserID, @WebinarID,@AvailableUntil);

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        ROLLBACK TRANSACTION;
        DECLARE @ErrorMessage nvarchar(4000) = ERROR_MESSAGE();
        DECLARE @ErrorSeverity int = ERROR_SEVERITY();
        DECLARE @ErrorState int = ERROR_STATE();
        RAISERROR (@ErrorMessage, @ErrorSeverity, @ErrorState);
    END CATCH
END;
```

6.0.106 Procedura p_CreateWebinar - Jakub Kaliński

Procedura p_CreateWebinar pozwala na utworzenie nowego webinaru w tabeli Webinars. Generuje unikalne WebinarID dla nowego webinaru i zapisuje przekazane dane w bazie.

Walidacja:

- Brak szczegółowej walidacji danych wejściowych w procedurze.
- Zaleca się weryfikację poprawności danych wejściowych po stronie aplikacji lub przed wywołaniem procedury.

```
CREATE OR ALTER PROCEDURE p_CreateWebinar
(
    @WebinarName VARCHAR(30),
    @TeacherID INT,
    @TranslatorID INT = NULL,
    @WebinarDate DATETIME,
    @Link VARCHAR(100),
    @DurationTime TIME(0) = NULL,
    @LinkToVideo VARCHAR(100),
    @WebinarDescription TEXT = NULL,
    @LanguageID INT = NULL,
  @Price MONEY
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        DECLARE @NewWebinarID INT;
```

```
22
23          SELECT @NewWebinarID = ISNULL(MAX(WebinarID), 0) + 1
24          FROM Webinars;
25
26      DECLARE @ServiceID INT;
27      EXEC p_AddService 'WebinarService', @ServiceID OUTPUT;
28      EXEC p_AddWebinarService @ServiceID, @Price;
29
30          INSERT INTO Webinars (
31              WebinarID, WebinarName, TeacherID, TranslatorID, WebinarDate,
32              Link, DurationTime, LinkToVideo, WebinarDescription,
33              LanguageID, ServiceID
34          ) VALUES (
35              @NewWebinarID, @WebinarName, @TeacherID, @TranslatorID, @WebinarDate,
36              @Link, @DurationTime, @LinkToVideo, @WebinarDescription,
37              @LanguageID, @ServiceID
38          );
39
40          COMMIT TRANSACTION;
41      END TRY
42      BEGIN CATCH
43          IF @@TRANCOUNT > 0
44              ROLLBACK TRANSACTION;
45
46          THROW;
47      END CATCH;
48  END;
```

6.0.107 Procedura p_EditWebinar - Jakub Kaliński

Procedura p_EditWebinar umożliwia aktualizację nazwy i daty wybranego webinaru.
Walidacja:

- Sprawdza, czy podane WebinarID istnieje w tabeli Webinars. Jeśli nie, wywołuje błąd.

```
1  CREATE OR ALTER PROCEDURE p_EditWebinar
2  (
3      @WebinarID INT,
4      @WebinarName VARCHAR(30) = NULL,
5      @WebinarDate DATETIME = NULL
6  )
7  AS
8  BEGIN
9      SET NOCOUNT ON;
10
11     BEGIN TRY
12         BEGIN TRANSACTION;
13
14         IF NOT EXISTS (SELECT 1 FROM Webinars WHERE WebinarID = @WebinarID)
15         BEGIN
16             RAISERROR('Invalid WebinarID: no matching Webinar found.', 16, 1);
17             ROLLBACK TRANSACTION;
18             RETURN;
19         END;
20
21         UPDATE Webinars
22         SET WebinarName = ISNULL(@WebinarName, WebinarName),
23             WebinarDate = ISNULL(@WebinarDate, WebinarDate)
24         WHERE WebinarID = @WebinarID;
25
26         COMMIT TRANSACTION;
27     END TRY
28     BEGIN CATCH
29         IF @@TRANCOUNT > 0
```

```
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.108 Procedura p_DeleteWebinar - Jakub Kaliński

Procedura p_DeleteWebinar umożliwia usunięcie webinaru oraz powiązanych z nim szczegółów z bazy danych.
Walidacja:

- Sprawdza, czy podane WebinarID istnieje w tabeli Webinars. Jeśli nie, wywołuje błąd.

```
CREATE OR ALTER PROCEDURE p_DeleteWebinar
(
    @WebinarID INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        -- Sprawdzenie, czy webinar istnieje
        IF NOT EXISTS (SELECT 1 FROM Webinars WHERE WebinarID = @WebinarID)
        BEGIN
            RAISERROR('Invalid WebinarID: no matching webinar found.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        -- Usuwanie powiązanych danych z tabeli WebinarDetails
        DELETE FROM WebinarDetails
        WHERE WebinarID = @WebinarID;

        -- Usuwanie webinaru z tabeli Webinars
        DELETE FROM Webinars
        WHERE WebinarID = @WebinarID;

        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.109 Procedura p_AddUser - Jakub Kaliński

Procedura p_AddUser dodaje nowego użytkownika do tabeli Users z przekazanymi danymi osobowymi.
Walidacja:

- Sprawdza, czy podana data urodzenia (DateOfBirth) nie jest przyszła. W przypadku niepoprawnej daty wywołuje błąd.

```sql
CREATE OR ALTER PROCEDURE p_AddUser
(
    @FirstName VARCHAR(30),
    @LastName VARCHAR(30),
    @DateOfBirth DATE = NULL,
    @UserTypeID INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        -- Walidacja daty urodzenia
        IF @DateOfBirth > GETDATE()
        BEGIN
            RAISERROR ('DateOfBirth cannot be in the future.', 16, 1);
            RETURN;
        END;

        -- Generowanie nowego UserID
        DECLARE @NewUserID INT;
        SELECT @NewUserID = ISNULL(MAX(UserID), 0) + 1 FROM Users;

        -- Wstawienie użytkownika do tabeli
        INSERT INTO Users (UserID, FirstName, LastName, DateOfBirth, UserTypeID)
        VALUES (@NewUserID, @FirstName, @LastName, @DateOfBirth, @UserTypeID);

    END TRY
    BEGIN CATCH
        THROW;
    END CATCH;
END;
```

6.0.110 Procedura p_UpdateUser - Jakub Kaliński

Procedura p_DeleteUser usuwa użytkownika z bazy danych wraz z powiązanymi danymi kontaktowymi i adresowymi.

Walidacja:

- Weryfikuje, czy podane UserID istnieje w tabeli. Jeśli nie, wywołuje błąd.

```sql
CREATE OR ALTER PROCEDURE p_UpdateUser
(
    @UserID INT,
    @FirstName VARCHAR(30) = NULL,
    @LastName VARCHAR(30) = NULL,
    @DateOfBirth DATE = NULL,
    @UserTypeID INT = NULL
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        IF @DateOfBirth IS NOT NULL AND @DateOfBirth > GETDATE()
        BEGIN
            RAISERROR('DateOfBirth cannot be in the future.', 16, 1);
            RETURN;
        END;

        UPDATE Users
        SET
            FirstName = COALESCE(@FirstName, FirstName),
```

```
            LastName = COALESCE(@LastName, LastName),
            DateOfBirth = COALESCE(@DateOfBirth, DateOfBirth),
            UserTypeID = COALESCE(@UserTypeID, UserTypeID)
        WHERE UserID = @UserID;

        IF @@ROWCOUNT = 0
        BEGIN
            RAISERROR('UserID not found.', 16, 1);
        END;
    END TRY
    BEGIN CATCH
        THROW;
    END CATCH;
END;
```

6.0.111 Procedura p_DeleteUser - Jakub Kaliński

```
CREATE OR ALTER PROCEDURE p_DeleteUser
(
    @UserID INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        -- Usuwanie zależnych danych w powiązanych tabelach
        DELETE FROM UserAddressDetails WHERE UserID = @UserID;
        DELETE FROM UserContact WHERE UserID = @UserID;

        -- Usuwanie użytkownika
        DELETE FROM Users WHERE UserID = @UserID;

        IF @@ROWCOUNT = 0
        BEGIN
            RAISERROR('UserID not found.', 16, 1);
        END;
    END TRY
    BEGIN CATCH
        THROW;
    END CATCH;
END;
```

6.0.112 Procedura p_AddEmployee - Jakub Kaliński

Procedura p_AddEmployee umożliwia dodanie nowego pracownika do tabeli Employees.
Walidacja:

- Sprawdza, czy data zatrudnienia (DateOfHire) nie jest przyszła. W przypadku niepoprawnej daty zgłasza błąd.

```
CREATE OR ALTER PROCEDURE p_AddEmployee
(
    @EmployeeID INT,
    @DateOfHire DATE = NULL
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        IF @DateOfHire IS NOT NULL AND @DateOfHire > GETDATE()
```

```
        BEGIN
            RAISERROR('DateOfHire cannot be in the future.', 16, 1);
            RETURN;
        END;

        INSERT INTO Employees (EmployeeID, DateOfHire)
        VALUES (@EmployeeID, @DateOfHire);
    END TRY
    BEGIN CATCH
        THROW;
    END CATCH;
END;
```

6.0.113 Procedura p_UpdateEmployee - Jakub Kaliński

Procedura p_UpdateEmployee pozwala zaktualizować datę zatrudnienia pracownika w tabeli Employees.
Walidacja:

- Sprawdza, czy podana data zatrudnienia (DateOfHire) nie jest przyszła.
- Weryfikuje, czy podane EmployeeID istnieje w tabeli. Jeśli nie, wywołuje błąd.

```
CREATE OR ALTER PROCEDURE p_UpdateEmployee
(
    @EmployeeID INT,
    @DateOfHire DATE = NULL
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        IF @DateOfHire IS NOT NULL AND @DateOfHire > GETDATE()
        BEGIN
            RAISERROR('DateOfHire cannot be in the future.', 16, 1);
            RETURN;
        END;

        UPDATE Employees
        SET DateOfHire = COALESCE(@DateOfHire, DateOfHire)
        WHERE EmployeeID = @EmployeeID;

        IF @@ROWCOUNT = 0
        BEGIN
            RAISERROR('EmployeeID not found.', 16, 1);
        END;
    END TRY
    BEGIN CATCH
        THROW;
    END CATCH;
END;
```

6.0.114 Procedura p_DeleteEmployee - Jakub Kaliński

Procedura p_DeleteEmployee usuwa pracownika z tabeli Employees oraz dane powiązane z innymi tabelami.
Walidacja:

- Sprawdza, czy podane EmployeeID istnieje w tabeli. Jeśli nie, zgłasza błąd.

```
CREATE OR ALTER PROCEDURE p_DeleteEmployee
(
    @EmployeeID INT
```

```
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        -- Usuwanie zależnych danych w powiązanych tabelach
        DELETE FROM EmployeesSuperior WHERE EmployeeID = @EmployeeID;
        DELETE FROM EmployeeDegree WHERE EmployeeID = @EmployeeID;

        -- Usuwanie pracownika
        DELETE FROM Employees WHERE EmployeeID = @EmployeeID;

        IF @@ROWCOUNT = 0
        BEGIN
            RAISERROR('EmployeeID not found.', 16, 1);
        END;
    END TRY
    BEGIN CATCH
        THROW;
    END CATCH;
END;
```

6.0.115 Procedura p_AssignSupervisor - Jakub Kaliński

Procedura p_AssignSupervisor pozwala przypisać przełożonego do danego pracownika w tabeli EmployeesSuperior.

Walidacja:

- Weryfikuje, czy podane SupervisorID istnieje w tabeli Employees. Jeśli nie, zgłasza błąd.

```
CREATE OR ALTER PROCEDURE p_AssignSupervisor
(
    @EmployeeID INT,
    @SupervisorID INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        -- Sprawdzenie, czy przełożony istnieje
        IF NOT EXISTS (SELECT 1 FROM Employees WHERE EmployeeID = @SupervisorID)
        BEGIN
            RAISERROR('SupervisorID not found.', 16, 1);
            RETURN;
        END;

        -- Aktualizacja tabeli przełożonych
        UPDATE EmployeesSuperior
        SET ReportsTo = @SupervisorID
        WHERE EmployeeID = @EmployeeID;

        IF @@ROWCOUNT = 0
        BEGIN
            INSERT INTO EmployeesSuperior (EmployeeID, ReportsTo)
            VALUES (@EmployeeID, @SupervisorID);
        END;
    END TRY
    BEGIN CATCH
        THROW;
    END CATCH;
END;
```

6.0.116 Procedura p_AddUserAddress - Jakub Kaliński

Procedura p_AddUserAddress dodaje adres użytkownika do tabeli UserAddressDetails.
Walidacja:

- Brak szczegółowej walidacji danych wejściowych w procedurze.

```
CREATE OR ALTER PROCEDURE p_AddUserAddress
(
    @UserID INT,
    @Address VARCHAR(30),
    @PostalCode VARCHAR(10),
    @LocationID INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        INSERT INTO UserAddressDetails (UserID, Address, PostalCode, LocationID)
        VALUES (@UserID, @Address, @PostalCode, @LocationID);
    END TRY
    BEGIN CATCH
        THROW;
    END CATCH;
END;
```

6.0.117 Procedura p_UpdateUserAddress - Jakub Kaliński

Procedura p_UpdateUserAddress pozwala zaktualizować dane adresowe użytkownika w tabeli UserAddressDetails.
Walidacja:

- Sprawdza, czy podane UserID istnieje w tabeli. Jeśli nie, wywołuje błąd.

```
CREATE OR ALTER PROCEDURE p_UpdateUserAddress
(
    @UserID INT,
    @Address VARCHAR(30) = NULL,
    @PostalCode VARCHAR(10) = NULL,
    @LocationID INT = NULL
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        UPDATE UserAddressDetails
        SET
            Address = COALESCE(@Address, Address),
            PostalCode = COALESCE(@PostalCode, PostalCode),
            LocationID = COALESCE(@LocationID, LocationID)
        WHERE UserID = @UserID;

        IF @@ROWCOUNT = 0
        BEGIN
            RAISERROR('UserID not found.', 16, 1);
        END;
    END TRY
    BEGIN CATCH
        THROW;
    END CATCH;
END;
```

6.0.118 Procedura p_DeleteUserAddress - Jakub Kaliński

Procedura p_DeleteUserAddress usuwa adres powiązany z użytkownikiem w tabeli UserAddressDetails.
Walidacja:

- Weryfikuje, czy podane UserID istnieje w tabeli. Jeśli nie, zgłasza błąd.

```
CREATE OR ALTER PROCEDURE p_DeleteUserAddress
(
    @UserID INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
    DELETE FROM UserAddressDetails WHERE UserID = @UserID;

IF @@ROWCOUNT = 0
BEGIN
RAISERROR('UserID not found.', 16, 1);
END;
END TRY
BEGIN CATCH
THROW;
END CATCH;
END;
```

6.0.119 Procedura p_AddUserContact - Jakub Kaliński

Procedura p_AddUserContact dodaje dane kontaktowe użytkownika do tabeli UserContact.
Walidacja:

- Sprawdza, czy numer telefonu (Phone) ma dokładnie 9 cyfr, jeśli jest podany. W przypadku błędnego formatu zgłasza błąd.

```
CREATE OR ALTER PROCEDURE p_AddUserContact
(
    @UserID INT,
    @Email VARCHAR(30),
    @Phone VARCHAR(30) = NULL
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        -- Walidacja numeru telefonu (jeśli nie jest NULL)
        IF @Phone IS NOT NULL AND NOT (@Phone LIKE
            '[1-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]')
        BEGIN
            RAISERROR('Phone number must be exactly 9 digits.', 16, 1);
            RETURN;
        END;

        INSERT INTO UserContact (UserID, Email, Phone)
        VALUES (@UserID, @Email, @Phone);
    END TRY
    BEGIN CATCH
        THROW;
    END CATCH;
END;
```

6.0.120 Procedura p_UpdateUserContact - Jakub Kaliński

Procedura p_UpdateUserContact pozwala na zaktualizowanie danych kontaktowych użytkownika w tabeli UserContact.
Walidacja:

- Sprawdza, czy numer telefonu (Phone) ma dokładnie 9 cyfr, jeśli jest podany.
- Weryfikuje, czy podane UserID istnieje w tabeli. Jeśli nie, zgłasza błąd.

```
CREATE OR ALTER PROCEDURE p_UpdateUserContact
(
    @UserID INT,
    @Email VARCHAR(30) = NULL,
    @Phone VARCHAR(30) = NULL
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        -- Walidacja numeru telefonu (jeśli nie jest NULL)
        IF @Phone IS NOT NULL AND NOT (@Phone LIKE
            '[1-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]')
        BEGIN
            RAISERROR('Phone number must be exactly 9 digits.', 16, 1);
            RETURN;
        END;

        UPDATE UserContact
        SET
            Email = COALESCE(@Email, Email),
            Phone = COALESCE(@Phone, Phone)
        WHERE UserID = @UserID;

        IF @@ROWCOUNT = 0
        BEGIN
            RAISERROR('UserID not found.', 16, 1);
        END;
    END TRY
    BEGIN CATCH
        THROW;
    END CATCH;
END;
```

6.0.121 Procedura p_DeleteUserContact - Jakub Kaliński

Procedura p_DeleteUserContact usuwa dane kontaktowe użytkownika z tabeli UserContact.
Walidacja:

- Sprawdza, czy podane UserID istnieje w tabeli. Jeśli nie, zgłasza błąd.

```
CREATE OR ALTER PROCEDURE p_DeleteUserContact
(
    @UserID INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        DELETE FROM UserContact WHERE UserID = @UserID;

        IF @@ROWCOUNT = 0
```

```
        BEGIN
            RAISERROR('UserID not found.', 16, 1);
        END;
    END TRY
    BEGIN CATCH
        THROW;
    END CATCH;
END;
```

6.0.122 Procedura p_AddUserType - Jakub Kaliński

Procedura p_AddUserType pozwala na dodanie nowego typu użytkownika do tabeli UserType.
Walidacja:

- Brak szczegółowej walidacji danych wejściowych w procedurze.

```
CREATE OR ALTER PROCEDURE p_AddUserType
(
    @UserTypeID INT,
    @UserTypeName VARCHAR(30)
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        INSERT INTO UserType (UserTypeID, UserTypeName)
        VALUES (@UserTypeID, @UserTypeName);
    END TRY
    BEGIN CATCH
        THROW;
    END CATCH;
END;
```

6.0.123 Procedura p_UpdateUserType - Jakub Kaliński

Procedura aktualizuje nazwę typu użytkownika w tabeli UserType, identyfikując rekord za pomocą UserTypeID. Jeśli podany UserTypeID nie istnieje, generowany jest błąd.
Walidacja:

- UserTypeID - powinno być wartością istniejącą w tabeli UserType. W przeciwnym razie procedura zwróci błąd.
- UserTypeName - nie powinno być puste ani przekraczać 30 znaków.

```
CREATE OR ALTER PROCEDURE p_UpdateUserType
(
    @UserTypeID INT,
    @UserTypeName VARCHAR(30)
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        UPDATE UserType
        SET UserTypeName = @UserTypeName
        WHERE UserTypeID = @UserTypeID;

        IF @@ROWCOUNT = 0
        BEGIN
            RAISERROR('UserTypeID not found.', 16, 1);
        END;
```

```
    END TRY
    BEGIN CATCH
        THROW;
    END CATCH;
END;
```

6.0.124 Procedura p_DeleteUserType - Jakub Kaliński

Procedura usuwa typ użytkownika z tabeli UserType na podstawie UserTypeID. Jeśli podany UserTypeID nie istnieje, generowany jest błąd.
Walidacja:

- UserTypeID - powinno być wartością istniejącą w tabeli UserType. W przeciwnym razie procedura zwróci błąd.

```
CREATE OR ALTER PROCEDURE p_DeleteUserType
(
    @UserTypeID INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        DELETE FROM UserType WHERE UserTypeID = @UserTypeID;

        IF @@ROWCOUNT = 0
        BEGIN
            RAISERROR('UserTypeID not found.', 16, 1);
        END;
    END TRY
    BEGIN CATCH
        THROW;
    END CATCH;
END;
```

6.0.125 Procedura p_AddDegree - Jakub Kaliński

Procedura dodaje nowy stopień naukowy do tabeli Degrees.
Walidacja:

- DegreeID - powinno być unikalne w tabeli Degrees.

DegreeLevel i DegreeName - nie powinny być puste ani przekraczać 30 znaków.

```
CREATE OR ALTER PROCEDURE p_AddDegree
(
    @DegreeLevel VARCHAR(30),
    @DegreeName VARCHAR(30)
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        -- Generowanie nowego UserID
        DECLARE @NewDegreeID INT;
        SELECT @NewDegreeID = ISNULL(MAX(UserID), 0) + 1 FROM Degrees;

        INSERT INTO Degrees (DegreeID, DegreeLevel, DegreeName)
        VALUES (@NewDegreeID, @DegreeLevel, @DegreeName);
    END TRY
    BEGIN CATCH
```

```
        THROW;
    END CATCH;
END;
```

6.0.126 Procedura p_UpdateDegree - Jakub Kaliński

Procedura aktualizuje informacje o stopniu naukowym w tabeli Degrees, identyfikując rekord za pomocą DegreeID. Jeśli podany DegreeID nie istnieje, generowany jest błąd.
Walidacja:

- DegreeID - powinno być wartością istniejącą w tabeli Degrees. W przeciwnym razie procedura zwróci błąd.

DegreeLevel i DegreeName - nie powinny być puste ani przekraczać 30 znaków.

```
CREATE OR ALTER PROCEDURE p_UpdateDegree
(
    @DegreeID INT,
    @DegreeLevel VARCHAR(30),
    @DegreeName VARCHAR(30)
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        UPDATE Degrees
        SET
            DegreeLevel = @DegreeLevel,
            DegreeName = @DegreeName
        WHERE DegreeID = @DegreeID;

        IF @@ROWCOUNT = 0
        BEGIN
            RAISERROR('DegreeID not found.', 16, 1);
        END;
    END TRY
    BEGIN CATCH
        THROW;
    END CATCH;
END;
```

6.0.127 Procedura p_DeleteDegree - Jakub Kaliński

Procedura usuwa stopień naukowy z tabeli Degrees na podstawie DegreeID. Jeśli podany DegreeID nie istnieje, generowany jest błąd.
Walidacja:

- DegreeID - powinno być wartością istniejącą w tabeli Degrees. W przeciwnym razie procedura zwróci błąd.

```
CREATE OR ALTER PROCEDURE p_DeleteDegree
(
    @DegreeID INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        DELETE FROM Degrees WHERE DegreeID = @DegreeID;

        IF @@ROWCOUNT = 0
```

```
        BEGIN
            RAISERROR('DegreeID not found.', 16, 1);
        END;
    END TRY
    BEGIN CATCH
        THROW;
    END CATCH;
END;
```

6.0.128 Procedura p_AddEmployeeDegree - Jakub Kaliński

Procedura przypisuje pracownikowi nowy stopień naukowy w tabeli EmployeeDegree.
Walidacja:

- EmployeeID - powinno istnieć w tabeli Employee.
- DegreeID - powinno istnieć w tabeli Degrees.

```
CREATE OR ALTER PROCEDURE p_AddEmployeeDegree
(
    @EmployeeID INT,
    @DegreeID INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        INSERT INTO EmployeeDegree (EmployeeID, DegreeID)
        VALUES (@EmployeeID, @DegreeID);
    END TRY
    BEGIN CATCH
        THROW;
    END CATCH;
END;
```

6.0.129 Procedura p_UpdateEmployeeDegree - Jakub Kaliński

Procedura aktualizuje przypisanie stopnia naukowego pracownika, zamieniając stary stopień na nowy w tabeli EmployeeDegree.
Walidacja:

- EmployeeID - powinno istnieć w tabeli Employee.
- OldDegreeID - powinno istnieć dla danego EmployeeID w tabeli EmployeeDegree. W przeciwnym razie procedura zwróci błąd.
- NewDegreeID - powinno istnieć w tabeli Degrees. W przeciwnym razie procedura zwróci błąd.

```
CREATE OR ALTER PROCEDURE p_UpdateEmployeeDegree
(
    @EmployeeID INT,
    @OldDegreeID INT,
    @NewDegreeID INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        -- Sprawdzenie, czy stary stopień istnieje dla pracownika
        IF NOT EXISTS (
            SELECT 1
```

```
            FROM EmployeeDegree
            WHERE EmployeeID = @EmployeeID AND DegreeID = @OldDegreeID
        )
        BEGIN
            RAISERROR('Old degree not found for the specified employee.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        -- Sprawdzenie, czy nowy stopień istnieje w tabeli Degrees
        IF NOT EXISTS (
            SELECT 1
            FROM Degrees
            WHERE DegreeID = @NewDegreeID
        )
        BEGIN
            RAISERROR('New degree does not exist.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        -- Aktualizacja stopnia naukowego pracownika
        UPDATE EmployeeDegree
        SET DegreeID = @NewDegreeID
        WHERE EmployeeID = @EmployeeID AND DegreeID = @OldDegreeID;

        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

6.0.130 Procedura p_DeleteEmployeeDegree - Jakub Kaliński

Procedura usuwa przypisanie stopnia naukowego z tabeli EmployeeDegree na podstawie EmployeeID i DegreeID.

Walidacja:

- EmployeeID i DegreeID - powinny istnieć w tabeli EmployeeDegree. W przeciwnym razie procedura zwróci błąd.

```
CREATE OR ALTER PROCEDURE p_DeleteEmployeeDegree
(
    @EmployeeID INT,
    @DegreeID INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        -- Sprawdzenie, czy wpis istnieje w tabeli EmployeeDegree
        IF NOT EXISTS (
            SELECT 1
            FROM EmployeeDegree
            WHERE EmployeeID = @EmployeeID AND DegreeID = @DegreeID
        )
```

```
        BEGIN
            RAISERROR('No matching record found in EmployeeDegree.', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END;

        -- Usunięcie wpisu z tabeli EmployeeDegree
        DELETE FROM EmployeeDegree
        WHERE EmployeeID = @EmployeeID AND DegreeID = @DegreeID;

        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
        IF @@TRANCOUNT > 0
        ROLLBACK TRANSACTION;

        -- Rzucenie błędu dalej
        THROW;
    END CATCH;
END;
```

6.0.131 Procedura p_AddTranslator - Jakub Kaliński

Procedura dodaje nowego tłumacza do tabeli Translators.
Walidacja:

- TranslatorID - powinno być unikalne w tabeli Translators.

```
CREATE OR ALTER PROCEDURE p_AddTranslator
(
    @TranslatorID INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        INSERT INTO Translators (TranslatorID)
        VALUES (@TranslatorID);
    END TRY
    BEGIN CATCH
        THROW;
    END CATCH;
END;
```

6.0.132 Procedura p_DeleteTranslator - Jakub Kaliński

Procedura usuwa tłumacza z tabeli Translators na podstawie TranslatorID. Jeśli TranslatorID nie istnieje, generowany jest błąd.
Walidacja:

- TranslatorID - powinno być wartością istniejącą w tabeli Translators. W przeciwnym razie procedura zwróci błąd.

```
CREATE OR ALTER PROCEDURE p_DeleteTranslator
(
    @TranslatorID INT
)
AS
BEGIN
    SET NOCOUNT ON;
```

```
8
9       BEGIN TRY
10          DELETE FROM Translators WHERE TranslatorID = @TranslatorID;
11
12          IF @@ROWCOUNT = 0
13          BEGIN
14              RAISERROR('TranslatorID not found.', 16, 1);
15          END;
16      END TRY
17      BEGIN CATCH
18          THROW;
19      END CATCH;
20  END;
```

6.0.133 Procedura p_AddLanguage - Jakub Kaliński

Procedura dodaje nowy język do tabeli Languages.
Walidacja:

- LanguageID - powinno być unikalne w tabeli Languages.
- LanguageName - nie powinno być puste ani przekraczać 30 znaków.

```
1   CREATE OR ALTER PROCEDURE p_AddLanguage
2   (
3       @LanguageName VARCHAR(30)
4   )
5   AS
6   BEGIN
7       SET NOCOUNT ON;
8
9       BEGIN TRY
10          DECLARE @NewLanguageID INT;
11
12          -- Ustalenie nowego ID jako największe istniejące +1
13          SELECT @NewLanguageID = ISNULL(MAX(LanguageID), 0) + 1 FROM Languages;
14          INSERT INTO Languages (LanguageID, LanguageName)
15          VALUES (@NewLanguageID, @LanguageName);
16      END TRY
17      BEGIN CATCH
18          THROW;
19      END CATCH;
20  END;
```

6.0.134 Procedura p_UpdateLanguage - Jakub Kaliński

Procedura aktualizuje informacje o języku w tabeli Languages, identyfikując rekord za pomocą LanguageID. Jeśli LanguageID nie istnieje, generowany jest błąd.
Walidacja:

- LanguageID - powinno być wartością istniejącą w tabeli Languages. W przeciwnym razie procedura zwróci błąd.
- LanguageName - nie powinno być puste ani przekraczać 30 znaków.

```
1   CREATE OR ALTER PROCEDURE p_UpdateLanguage
2   (
3       @LanguageID INT,
4       @LanguageName VARCHAR(30)
5   )
6   AS
7   BEGIN
8       SET NOCOUNT ON;
9
```

```
    BEGIN TRY
        UPDATE Languages
        SET LanguageName = @LanguageName
        WHERE LanguageID = @LanguageID;

        IF @@ROWCOUNT = 0
        BEGIN
            RAISERROR('LanguageID not found.', 16, 1);
        END;
    END TRY
    BEGIN CATCH
        THROW;
    END CATCH;
END;
```

6.0.135 Procedura p_DeleteLanguage - Jakub Kaliński

Procedura usuwa język z tabeli Languages na podstawie LanguageID. Jeśli LanguageID nie istnieje, generowany jest błąd.
Walidacja:

- LanguageID - powinno być wartością istniejącą w tabeli Languages. W przeciwnym razie procedura zwróci błąd.

```
CREATE OR ALTER PROCEDURE p_DeleteLanguage
(
    @LanguageID INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        DELETE FROM Languages WHERE LanguageID = @LanguageID;

        IF @@ROWCOUNT = 0
        BEGIN
            RAISERROR('LanguageID not found.', 16, 1);
        END;
    END TRY
    BEGIN CATCH
        THROW;
    END CATCH;
END;
```

6.0.136 Procedura p_AddTranslatorLanguage - Jakub Kaliński

Procedura przypisuje język tłumaczowi w tabeli TranslatorsLanguages.
Walidacja:

- TranslatorID - powinno istnieć w tabeli Translators.
- LanguageID - powinno istnieć w tabeli Languages.

```
CREATE OR ALTER PROCEDURE p_AddTranslatorLanguage
(
    @TranslatorID INT,
    @LanguageID INT
)
AS
BEGIN
    SET NOCOUNT ON;

```

```
    BEGIN TRY
        INSERT INTO TranslatorsLanguages (TranslatorID, LanguageID)
        VALUES (@TranslatorID, @LanguageID);
    END TRY
    BEGIN CATCH
        THROW;
    END CATCH;
END;
```

6.0.137 Procedura p_DeleteTranslatorLanguage - Jakub Kaliński

Procedura usuwa przypisanie języka tłumaczowi z tabeli TranslatorsLanguages na podstawie TranslatorID i LanguageID. Jeśli TranslatorID lub LanguageID nie istnieje, generowany jest błąd.
Walidacja:

- TranslatorID i LanguageID - powinny istnieć w tabeli TranslatorsLanguages. W przeciwnym razie procedura zwróci błąd.

```
CREATE OR ALTER PROCEDURE p_DeleteTranslatorLanguage
(
    @TranslatorID INT,
    @LanguageID INT
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        DELETE FROM TranslatorsLanguages
        WHERE TranslatorID = @TranslatorID AND LanguageID = @LanguageID;

        IF @@ROWCOUNT = 0
        BEGIN
            RAISERROR('TranslatorID and/or LanguageID not found.', 16, 1);
        END;
    END TRY
    BEGIN CATCH
        THROW;
    END CATCH;
END;
```

6.0.138 Procedura p_AddLocation - Jakub Kaliński

Procedura dodaje nową lokalizację do tabeli Locations.
Walidacja:

- LocationID - powinno być unikalne w tabeli Locations.
- CountryName, ProvinceName i CityName - nie powinny przekraczać odpowiednio 30 i 50 znaków.

```
CREATE OR ALTER PROCEDURE p_AddLocation
(
    @LocationID INT,
    @CountryName VARCHAR(30),
    @ProvinceName VARCHAR(50) = NULL,
    @CityName VARCHAR(50)
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        INSERT INTO Locations (LocationID, CountryName, ProvinceName, CityName)
```

```
        VALUES (@LocationID, @CountryName, @ProvinceName, @CityName);
    END TRY
    BEGIN CATCH
        THROW;
    END CATCH;
END;
```

6.0.139 Procedura p_UpdateLocation - Jakub Kaliński

Procedura aktualizuje informacje o lokalizacji w tabeli Locations, identyfikując rekord za pomocą LocationID. Jeśli LocationID nie istnieje, generowany jest błąd.

Walidacja:

- LocationID - powinno być wartością istniejącą w tabeli Locations. W przeciwnym razie procedura zwróci błąd.
- CountryName, ProvinceName i CityName - nie powinny przekraczać odpowiednio 30 i 50 znaków.

```
CREATE OR ALTER PROCEDURE p_UpdateLocation
(
    @LocationID INT,
    @CountryName VARCHAR(30) = NULL,
    @ProvinceName VARCHAR(50) = NULL,
    @CityName VARCHAR(50) = NULL
)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        UPDATE Locations
        SET
            CountryName = COALESCE(@CountryName, CountryName),
            ProvinceName = COALESCE(@ProvinceName, ProvinceName),
            CityName = COALESCE(@CityName, CityName)
        WHERE LocationID = @LocationID;

        IF @@ROWCOUNT = 0
        BEGIN
            RAISERROR('LocationID not found.', 16, 1);
        END;
    END TRY
    BEGIN CATCH
        THROW;
    END CATCH;
END;
```

6.0.140 Procedura p_DeleteLocation - Jakub Kaliński

Procedura usuwa lokalizację z tabeli Locations na podstawie LocationID. Jeśli LocationID nie istnieje, generowany jest błąd.

Walidacja:

- LocationID - powinno być wartością istniejącą w tabeli Locations. W przeciwnym razie procedura zwróci błąd.

```
CREATE OR ALTER PROCEDURE p_DeleteLocation
(
    @LocationID INT
)
AS
BEGIN
    SET NOCOUNT ON;
```

```
8
9       BEGIN TRY
10          DELETE FROM Locations WHERE LocationID = @LocationID;
11
12          IF @@ROWCOUNT = 0
13          BEGIN
14              RAISERROR('LocationID not found.', 16, 1);
15          END;
16      END TRY
17      BEGIN CATCH
18          THROW;
19      END CATCH;
20  END;
```

7 Funkcje

7.0.1 Funkcja f_CalculateAttendancePercentageOnModule - Emil Żychowicz

Opis:
Oblicza procent obecności uczestnika na wszystkich spotkaniach w module danego kursu (zarówno spotkaniach stacjonarnych, jak i nagraniach). Jeśli uczestnik lub kurs nie istnieje, zwraca `0.0`.

Zastosowanie :
Alternatywne podejście do obliczeń w widoku zaliczeń kursu. Użyte w `f_CheckIfCourseIsPassed`.

Zwraca:
`FLOAT` – procent obecności uczestnika na module (wartość od `0.0` do `100.0`).

```
1   CREATE OR ALTER FUNCTION f_CalculateAttendancePercentageOnModule(
2       @ParticipantID INT,
3       @CourseID INT,
4       @ModuleID INT
5   )
6   RETURNS FLOAT
7   AS
8   BEGIN
9       DECLARE @TotalSessions INT;
10      DECLARE @AttendedSessions INT;
11      DECLARE @AttendancePercentage FLOAT;
12
13    IF NOT EXISTS (SELECT * FROM CourseParticipants WHERE ParticipantID = @ParticipantID)
          OR NOT EXISTS (SELECT * FROM Courses WHERE CourseID = @CourseID)
14      BEGIN
15          RETURN 0.0;
16      END
17
18    -- policz liczbę zajęć
19      SELECT @TotalSessions = COUNT(*)
20      FROM ATTENDANCE_MEETINGS_IN_COURSES
21    WHERE CourseID = @CourseID AND ModuleID = @ModuleID;
22
23    -- policz liczbę zajęć-nagrań
24    SELECT @TotalSessions += COUNT(DISTINCT MeetingID)
25    FROM ATTENDANCE_LISTS_OFFLINEVIDEO_COURSES
26    WHERE CourseID = @CourseID AND ModuleID = @ModuleID;
27
28    -- zlicz obecności
29      SELECT @AttendedSessions = SUM(CAST(Attendance AS INT))
30      FROM ATTENDANCE_LISTS_COURSES
31      WHERE CourseID = @CourseID AND ModuleID = @ModuleID AND ParticipantID =
            @ParticipantID;
32
33    SELECT @AttendedSessions += SUM(CAST(Attendance AS INT))
34    FROM ATTENDANCE_LISTS_OFFLINEVIDEO_COURSES
35    WHERE CourseID = @CourseID AND ModuleID = @ModuleID AND ParticipantID = @ParticipantID;
```

```
36
37      IF @TotalSessions > 0
38          SET @AttendancePercentage = CAST(@AttendedSessions AS FLOAT) / @TotalSessions *
                100;
39      ELSE
40          SET @AttendancePercentage = 0;
41
42      RETURN @AttendancePercentage;
43  END
```

7.0.2 Funkcja f_CalculateMINRoomCapacityCourse - Emil Żychowicz

Opis:
Oblicza maksymalną minimalną pojemność sal używanych dla kursu stacjonarnego. Jeśli kurs nie ma spotkań stacjonarnych, zwraca maksymalną wartość `INT` (2147483647). Jeśli nie istnieją odpowiednie dane, zwraca `0`.

Zastosowanie: Sprawdzanie, czy określone przyporządkowanie pokojów do kursów jest fizycznie akceptowalne.

Zwraca:
`INT` – minimalną pojemność sal (lub specjalną wartość `2147483647` dla kursów bez spotkań stacjonarnych).

```
1  CREATE OR ALTER FUNCTION f_CalculateMINRoomCapacityCourse(@CourseID INT)
2  RETURNS INT
3  AS
4  BEGIN
5    DECLARE @Result INT;
6    IF NOT EXISTS (SELECT * FROM COURSE_INFO WHERE CourseID = @CourseID AND MeetingType =
         'Stationary')
7    BEGIN
8      SET @Result = 2147483647;
9    END;
10
11   SELECT @Result = MAX(T1.Capacity)
12   FROM
13   (SELECT r.Capacity
14   FROM Rooms as r
15   INNER JOIN StationaryMeeting as sm
16   on sm.RoomID = r.RoomID
17   WHERE sm.MeetingID IN (SELECT MeetingID FROM COURSE_INFO WHERE CourseID = @CourseID))
         as T1
18
19   RETURN ISNULL(@Result, 0);
20 END;
```

7.0.3 Funkcja f_CalculateOrderValue - Emil Żychowicz

Opis:
Oblicza całkowitą wartość zamówienia użytkownika na podstawie cen usług, biorąc pod uwagę różne typy usług. W przypadku usług `ClassMeetingService` uwzględnia zniżki dla studentów, jeśli mają zastosowanie.

Zwraca:
`FLOAT` – wartość zamówienia w walucie systemowej.

```
1  CREATE OR ALTER FUNCTION f_CalculateOrderValue(@ParticipantID INT, @OrderID INT)
2  RETURNS FLOAT
3  AS
4  BEGIN
5      DECLARE @Value FLOAT = 0;
6      DECLARE @SumValue FLOAT = 0;
7
8      SELECT @Value = COALESCE(SUM(CAST(fp.FullPrice AS FLOAT)), 0)
9      FROM OrderDetails AS od
10     INNER JOIN FULL_PRICE AS fp
11         ON od.ServiceID = fp.ServiceID
```

```
    WHERE (SELECT ServiceType FROM Services WHERE ServiceID = od.ServiceID) !=
        'ClassMeetingService'
      AND od.OrderID = @OrderID;

    -- Obsługa ClassMeetingService w zależności od typu użytkownika
    IF NOT EXISTS (SELECT * FROM STUDIES_USERS WHERE @ParticipantID = ServiceUserID)
    BEGIN
        SELECT @SumValue = COALESCE(SUM(CAST(fp.FullPrice AS FLOAT)), 0)
        FROM OrderDetails AS od
        INNER JOIN FULL_PRICE AS fp
            ON od.ServiceID = fp.ServiceID
        WHERE (SELECT ServiceType FROM Services WHERE ServiceID = od.ServiceID) =
            'ClassMeetingService'
          AND od.OrderID = @OrderID;
    END
    ELSE
    BEGIN
        SELECT @SumValue = COALESCE(SUM(CASE
                                        WHEN cms.PriceStudents IS NOT NULL THEN
                                            CAST(cms.PriceStudents AS FLOAT)
                                        ELSE CAST(cms.PriceOthers AS FLOAT)
                                    END), 0)
        FROM OrderDetails AS od
        INNER JOIN ClassMeetingService AS cms
            ON od.ServiceID = cms.ServiceID
        WHERE (SELECT ServiceType FROM Services WHERE ServiceID = od.ServiceID) =
            'ClassMeetingService'
          AND od.OrderID = @OrderID;
    END;
    SET @Value += @SumValue;

    RETURN @Value;
END;
```

7.0.4 Funkcja f_CalculatePaidOrderValue - Emil Żychowicz

Opis:
Oblicza całkowitą wartość płatności przypisanych do danego zamówienia.
Zwraca:
`MONEY` – suma wszystkich wpłaconych kwot dla zamówienia.

```
CREATE OR ALTER FUNCTION f_CalculatePaidOrderValue(@OrderID INT)
RETURNS MONEY
AS
BEGIN
  DECLARE @Value MONEY;
  SELECT @Value = SUM(PaymentValue)
  FROM Payments
  WHERE OrderID = @OrderID
  RETURN ISNULL(@Value,0);
END;
```

7.0.5 Funkcja f_CalculatePaidServiceValue - Emil Żychowicz

Opis:
Oblicza całkowitą wartość wpłat użytkownika dla konkretnej usługi w ramach zamówienia.
Zastosowanie:
Używane przy sprawdzaniu stanu opłacenia usługi (`f_IsReadyToParticipate`). **Zwraca:**
`FLOAT` – suma wpłaconych kwot przypisanych do danej usługi.

```
CREATE OR ALTER FUNCTION f_CalculatePaidServiceValue(@ServiceID INT, @OrderID INT)
RETURNS FLOAT
AS
```

```
BEGIN
    DECLARE @PaidValue FLOAT;

    SELECT @PaidValue = COALESCE(SUM(CAST(PaymentValue AS FLOAT)), 0)
    FROM Payments
    WHERE ServiceID = @ServiceID
      AND OrderID = @OrderID;

    RETURN @PaidValue;
END;
```

7.0.6 Funkcja f_CheckIfCourseIsPassed - Emil Żychowicz

Opis:
Sprawdza, czy uczestnik ukończył kurs. Uczestnik musi uzyskać 100% obecności na co najmniej 80% modułów kursu, aby kurs został uznany za ukończony.

Zastosowanie:
Alternatywne podejście do sprawdzania czy użytkownik zdał w widoku `COURSE_PASSING_STATUS`.

Zwraca:
`BIT` – `1`, jeśli kurs jest ukończony; `0` w przeciwnym razie.

```
CREATE OR ALTER FUNCTION f_CheckIfCourseIsPassed(
    @ParticipantID INT,
    @CourseID INT
)
RETURNS BIT
AS
BEGIN
    DECLARE @ModuleCount INT;
  DECLARE @PassedModules INT;
  DECLARE @Result BIT;
  SELECT @ModuleCount = COUNT(*)
  FROM Modules
  WHERE CourseID = @CourseID;

  IF @ModuleCount = 0 --gdy kurs pusty
  BEGIN
    RETURN 0;
  END;

  SELECT @PassedModules = COUNT(*)
  FROM Modules
  Where CourseID = @CourseID AND
      ROUND(dbo.f_CalculateAttendancePercentageOnModule(@ParticipantID, @CourseID,
      ModuleID),0) = 100
  IF ROUND(@PassedModules* 1.0 / @ModuleCount,2) > 0.8
  BEGIN
    SET @Result = 1;
  END
  ELSE
  BEGIN
    SET @Result = 0;
  END
  RETURN @Result;
END;
```

7.0.7 Funkcja f_CourseSchedule - Emil Żychowicz

Opis:
Zwraca harmonogram spotkań dla kursu (zarówno stacjonarnych, jak i online) w formie tabelarycznej, z informacjami o dacie rozpoczęcia i zakończenia spotkania.

Zwraca:
`TABLE` – tabela zawierająca kolumny: `ModuleID`, `MeetingID`, `MeetingType`, `StartOfMeeting`, `EndOfMeeting`.

```
CREATE OR ALTER FUNCTION f_CourseSchedule(@CourseID INT)
RETURNS TABLE
AS
RETURN
    WITH CombinedMeetings AS (
        SELECT
      MeetingID,
            MeetingDate,
            MeetingDuration
        FROM StationaryMeeting
        UNION ALL
        SELECT
      MeetingID,
            MeetingDate,
            MeetingDuration
        FROM OnlineLiveMeeting
    )
    SELECT
        CI.ModuleID,
        CI.MeetingID,
        CI.MeetingType,
        CM.MeetingDate AS StartOfMeeting,
        DATEADD(MINUTE, DATEDIFF(MINUTE, '00:00:00', CM.MeetingDuration),
            CM.MeetingDate) AS EndOfMeeting
    FROM COURSE_INFO CI
    INNER JOIN CombinedMeetings CM
        ON CI.MeetingID = CM.MeetingID
    WHERE CI.CourseID = @CourseID
      AND CI.MeetingType != 'Offline Video';
```

7.0.8 Funkcja f_GetServiceValue - Emil Żychowicz

Opis:
Oblicza wartość usługi dla użytkownika. Dla usług `ClassMeetingService` uwzględnia różne ceny dla studentów i innych użytkowników.

Zastosowanie: Służy do obliczania stanu opłacenia danej usługi w `f_IsReadyToParticipate`.

Zwraca:
`FLOAT` – wartość usługi dla użytkownika.

```
CREATE OR ALTER FUNCTION f_GetServiceValue(@UserID INT, @ServiceID INT)
RETURNS FLOAT
AS
BEGIN
    DECLARE @Value FLOAT = 0;
    DECLARE @ServiceType NVARCHAR(50);

    SELECT @ServiceType = ServiceType
    FROM Services
    WHERE ServiceID = @ServiceID;

    IF @ServiceType = 'ClassMeetingService'
    BEGIN
        IF EXISTS (SELECT 1 FROM STUDIES_USERS WHERE ServiceUserID = @UserID)
        BEGIN
            SELECT @Value = COALESCE(PriceStudents, PriceOthers)
            FROM ClassMeetingService
            WHERE ServiceID = @ServiceID;
        END
        ELSE
        BEGIN
            SELECT @Value = COALESCE(PriceOthers, 0)
            FROM ClassMeetingService
```

```
            WHERE ServiceID = @ServiceID;
        END
    END
    ELSE
    BEGIN
        SELECT @Value = COALESCE(FullPrice, 0)
        FROM FULL_PRICE
        WHERE ServiceID = @ServiceID;
    END;

    RETURN @Value;
END;
```

7.0.9 Funkcja f_IsReadyToParticipate - Emil Żychowicz

Opis:
Sprawdza, czy użytkownik spełnia warunki uczestnictwa w danej usłudze. Uwzględnia:

- Zgody dyrektora,
- Pełne opłacenie usług (kursów, webinarów, itp.),
- Minimalne wymagania wpłat (`EntryFee`) dla studiów.
-

Zastosowanie: W widoku `CONSUMER_BASKET` i W triggerze `trg_AddPayment`, który sprawdzając czy usługa jest już w pełni opłacona, dodaje użytkownika do odpowiednich spotkań.

Zwraca:
`BIT` – `1`, jeśli użytkownik jest gotowy do uczestnictwa; `0` w przeciwnym razie.

```
CREATE OR ALTER FUNCTION f_IsReadyToParticipate(@UserID INT, @OrderID INT, @ServiceID
    INT)
RETURNS BIT
AS
BEGIN
  -- zgoda dyrektora: automatycznie gotowy,
  -- webinar, kurs, classmeeting, convention: opłacona w pełni kwota,
  -- studia: oplacone EntryFee
  IF (SELECT PrincipalAgreement FROM OrderDetails WHERE OrderID = @OrderID AND
      @ServiceID = ServiceID) = 1
  BEGIN
    RETURN 1;
  END;

  DECLARE @ServiceType VARCHAR(50);
  SET @ServiceType = (SELECT ServiceType FROM Services WHERE ServiceID = @ServiceID);

  IF @ServiceType IN ('ClassMeetingService', 'ConventionService', 'CourseService',
      'WebinarService')
    AND dbo.f_GetServiceValue(@UserID, @ServiceID) <=
        dbo.f_CalculatePaidServiceValue(@ServiceID, @OrderID)
  BEGIN
    RETURN 1;
  END;
  ELSE -- studia
  BEGIN
    IF dbo.f_CalculatePaidServiceValue(@ServiceID, @OrderID) >= (SELECT EntryFee FROM
        StudiesService WHERE ServiceID = @ServiceID)
    BEGIN
      RETURN 1;
    END;
  END
  RETURN 0;
END;
```

7.0.10 Funkcja f_CalculateAverageUserAge - Jakub Kaliński

```
CREATE OR ALTER FUNCTION f_CalculateAverageUserAge()
RETURNS FLOAT
AS
BEGIN
    DECLARE @AverageAge FLOAT;

    SET @AverageAge = (
        SELECT AVG(DATEDIFF(YEAR, DateOfBirth, GETDATE()))
        FROM Users
        WHERE DateOfBirth IS NOT NULL
    );

    RETURN @AverageAge;
END;
```

7.0.11 Funkcja f_CountEmployeesUnderSupervisor - Jakub Kaliński

```
CREATE OR ALTER FUNCTION f_CountEmployeesUnderSupervisor
(
    @SupervisorID INT
)
RETURNS INT
AS
BEGIN
    DECLARE @EmployeeCount INT;

    SET @EmployeeCount = (
        SELECT COUNT(*)
        FROM EmployeesSuperior
        WHERE ReportsTo = @SupervisorID
    );

    RETURN @EmployeeCount;
END;
```

7.0.12 Funkcja f_HasUserAddress - Jakub Kaliński

```
CREATE OR ALTER FUNCTION f_HasUserAddress
(
    @UserID INT
)
RETURNS BIT
AS
BEGIN
    DECLARE @HasAddress BIT;

    SET @HasAddress = CASE
        WHEN EXISTS (
            SELECT 1
            FROM UserAddressDetails
            WHERE UserID = @UserID
        ) THEN 1
        ELSE 0
    END;

    RETURN @HasAddress;
END;
```

7.0.13 Funkcja f_CountTranslatorLanguages - Jakub Kaliński

```
CREATE OR ALTER FUNCTION f_CountTranslatorLanguages
(
    @TranslatorID INT
)
RETURNS INT
AS
BEGIN
    DECLARE @LanguageCount INT;

    SET @LanguageCount = (
        SELECT COUNT(*)
        FROM TranslatorsLanguages
        WHERE TranslatorID = @TranslatorID
    );

    RETURN @LanguageCount;
END;
```

7.0.14 Funkcja f_CountWebinarParticipants - Jakub Kaliński

```
CREATE OR ALTER FUNCTION f_CountWebinarParticipants
(
    @WebinarID INT
)
RETURNS INT
AS
BEGIN
    DECLARE @ParticipantCount INT;

    SET @ParticipantCount = (
        SELECT COUNT(*)
        FROM WebinarDetails
        WHERE WebinarID = @WebinarID
    );

    RETURN @ParticipantCount;
END;
```

7.0.15 Funkcja f_IsUserRegisteredForWebinar - Jakub Kaliński

```
CREATE OR ALTER FUNCTION f_IsUserRegisteredForWebinar
(
    @UserID INT,
    @WebinarID INT
)
RETURNS BIT
AS
BEGIN
    DECLARE @IsRegistered BIT;

    SET @IsRegistered = CASE
        WHEN EXISTS (
            SELECT 1
            FROM WebinarDetails
            WHERE UserID = @UserID AND WebinarID = @WebinarID
        ) THEN 1
        ELSE 0
    END;

    RETURN @IsRegistered;
```

```
END;
```

7.0.16 Funkcja f_IsUserEmployee - Jakub Kaliński

```
CREATE OR ALTER FUNCTION f_IsUserEmployee
(
    @UserID INT
)
RETURNS BIT
AS
BEGIN
    DECLARE @IsEmployee BIT;

    SET @IsEmployee = CASE
        WHEN EXISTS (
            SELECT 1
            FROM Employees
            WHERE EmployeeID = @UserID
        ) THEN 1
        ELSE 0
    END;

    RETURN @IsEmployee;
END;
```

ł.md

7.0.17 Funkcja p_CalculateSubjectAttendance - Michał Szymocha

Oblicza procent obecności studenta na zajęciach z określonego przedmiotu, porównując liczbę uczęszczonych zajęć do całkowitej liczby zajęć.

```
Create or alter FUNCTION p_CalculateSubjectAttendance
(
    @SubjectID INT,
    @StudentID INT
)
returns FLOAT
AS
BEGIN
    DECLARE @TotalClasses INT;
    DECLARE @AttendedClasses INT;
    DECLARE @Attendance FLOAT;

    SET @TotalClasses = (SELECT COUNT(*)
    FROM SyncClassDetails
    WHERE MeetingID IN (SELECT ClassMeetingID
        FROM ClassMeeting
        WHERE SubjectID = @SubjectID) AND StudentID = @StudentID);
    SET @AttendedClasses = (SELECT COUNT(*)
    FROM SyncClassDetails
    WHERE Attendance = 1 and MeetingID IN (SELECT ClassMeetingID
        FROM ClassMeeting
        WHERE SubjectID = @SubjectID) AND StudentID = @StudentID);

    IF @TotalClasses = 0
    BEGIN
        SET @Attendance = 0;
    END
    ELSE
    BEGIN
        SET @Attendance = (@AttendedClasses * 100.0) / @TotalClasses;
    END

```

```
    RETURN @Attendance;
END;
```

7.0.18 Funkcja CalculateAvailableSeatsStudies - Michał Szymocha

Funkcja `CalculateAvailableSeatsStudies` zwraca liczbę wolnych miejsc na konkretnych studiach.

```
CREATE OR ALTER FUNCTION CalculateAvailableSeatsStudies
(
    @StudiesID INT
)
RETURNS INT
AS
BEGIN
    DECLARE @AvailableSeats INT;

    SET @AvailableSeats = (
        SELECT Studies.EnrollmentLimit - COUNT(*)
        FROM Studies
        JOIN StudiesDetails ON Studies.StudiesID = StudiesDetails.StudiesID
        WHERE Studies.StudiesID = @StudiesID
        GROUP BY Studies.EnrollmentLimit
    );

    RETURN @AvailableSeats;
END;
```

7.0.19 Funkcja p_CalculateStudiesAttendance - Michał Szymocha

Oblicza procent obecności studenta na zajęciach w ramach danego kierunku studiów, sprawdzając liczbę zajęć, na które student uczęszczał, w porównaniu do całkowitej liczby zajęć.

```
Create or alter FUNCTION p_CalculateStudiesAttendance
(
    @StudiesID INT,
    @StudentID INT
)
returns FLOAT
AS
BEGIN
    DECLARE @TotalClasses INT;
    DECLARE @AttendedClasses INT;
    DECLARE @Attendance FLOAT;

    IF NOT EXISTS (SELECT 1 FROM Studies WHERE StudiesID = @StudiesID)
    BEGIN
        RETURN 0.0;
    END;

    IF NOT EXISTS (SELECT 1 FROM Users WHERE UserID = @StudentID and UserTypeID = 1)
    BEGIN
        RETURN 0.0;
    END;

    IF NOT EXISTS (SELECT 1 FROM StudiesDetails WHERE StudiesID = @StudiesID and
        StudentID = @StudentID)
    BEGIN
        RETURN 0.0;
    END;

    SET @TotalClasses = (SELECT COUNT(*)
    FROM SyncClassDetails
    WHERE MeetingID IN (SELECT ClassMeetingID
        FROM ClassMeeting
```

```
        WHERE SubjectID IN (SELECT SubjectID
            FROM SubjectStudiesAssignment
            WHERE StudiesID = @StudiesID)) AND StudentID = @StudentID);
    SET @AttendedClasses = (SELECT COUNT(*)
    FROM SyncClassDetails
    WHERE Attendance = 1 and MeetingID IN (SELECT ClassMeetingID
        FROM ClassMeeting
        WHERE SubjectID IN (SELECT SubjectID
            FROM SubjectStudiesAssignment
            WHERE StudiesID = @StudiesID)) AND StudentID = @StudentID);

    IF @TotalClasses = 0
    BEGIN
        SET @Attendance = 0;
    END
    ELSE
    BEGIN
        SET @Attendance = (@AttendedClasses * 100.0) / @TotalClasses;
    END

    RETURN @Attendance;
END;

use u_szymocha
SELECT dbo.p_CalculateStudiesAttendance(1, 4) AS AttendancePercentage;
```

7.0.20 Funkcja p_CalculateInternshipCompletion - Michał Szymocha

Oblicza procent ukończenia stażu przez studentów w ramach konkretnego stażu, porównując liczbę studentów, którzy ukończyli staż, do wszystkich zapisanych na ten staż.

```
Create or alter function p_CalculateInternshipCompletion
(
    @InternshipID INT,
    @StudiesID INT
)
returns FLOAT
AS
BEGIN
    DECLARE @TotalStudents INT;
    DECLARE @CompletedStudents INT;
    DECLARE @Completion FLOAT;

    SET @TotalStudents = (SELECT COUNT(*)
    FROM InternshipDetails
    WHERE InternshipID = @InternshipID);
    SET @CompletedStudents = (SELECT COUNT(*)
    FROM InternshipDetails
    WHERE InternshipAttendance = 1 and InternshipID = @InternshipID);

    IF @TotalStudents = 0
    BEGIN
        SET @Completion = 0;
    END
    ELSE
    BEGIN
        SET @Completion = (@CompletedStudents * 100.0) / @TotalStudents;
    END

    RETURN @Completion;
END;
```

7.0.21 Funkcja p_CalculateAverageNumberOfPeopleInClass - Michał Szymocha

Oblicza średnią liczbę osób obecnych na zajęciach w ramach danego kierunku studiów, porównując liczbę obecnych studentów do liczby przeprowadzonych zajęć.

```
Create or alter function p_CalculateAverageNumberOfPeopleInClass
(
    @StudiesID INT
)
returns FLOAT
AS
BEGIN
    DECLARE @TotalClasses INT;
    DECLARE @TotalStudents INT;
    DECLARE @Average FLOAT;

    SET @TotalClasses = (SELECT COUNT(*)
    FROM ClassMeeting
    WHERE MeetingType in ('Stationary', 'OnlineLive') and SubjectID IN (SELECT SubjectID
            FROM SubjectStudiesAssignment
            WHERE StudiesID = @StudiesID));
    SET @TotalStudents = (SELECT COUNT(*)
    FROM SyncClassDetails
    WHERE Attendance = 1 and MeetingID IN (SELECT ClassMeetingID
        FROM ClassMeeting
        WHERE SubjectID IN (SELECT SubjectID
            FROM SubjectStudiesAssignment
            WHERE StudiesID = @StudiesID)));

    IF @TotalClasses = 0
    BEGIN
        SET @Average = 0;
    END
    ELSE
    BEGIN
        SET @Average = @TotalStudents / @TotalClasses;
    END

    RETURN @Average;
END;
```

7.0.22 Funkcja p_CalculateMINRoomCapacity - Michał Szymocha

Oblicza minimalną pojemność sali, w której odbywają się zajęcia z przedmiotu przypisanego do danego kierunku studiów.

```
CREATE or ALTER FUNCTION p_CalculateMINRoomCapacity
(
    @StudiesID INT
)
RETURNS INT
AS
BEGIN
    DECLARE @MINCapacity INT;

    SET @MINCapacity = (SELECT MIN(Capacity)
    FROM Rooms
    where RoomID in (Select RoomID
    from StationaryClass
    WHERE MeetingID IN (SELECT ClassMeetingID
    FROM ClassMeeting
    where SubjectID IN (SELECT SubjectID
    from SubjectStudiesAssignment
    WHERE StudiesID = @StudiesID))));
    RETURN @MINCapacity;
```

```
END;
```

7.0.23 Funkcja totalTimeSpentInClass - Michał Szymocha

Funkcja zwraca czas jaki student spędził na zajęciach stacjonarnych.

```
Create or alter FUNCTION totalTimeSpentInClass
(
    @UserID INT
)
RETURNS TIME
AS
BEGIN
    DECLARE @TotalTime TIME;

    SET @TotalTime = (
        SELECT SUM(DurationTime)
        FROM SyncClassDetails scd JOIN SyncClass sc ON scd.ClassID = sc.ClassID
        WHERE scd.UserID = @UserID and scd.Attendance = 1
    );

    RETURN @TotalTime;
END;
```

7.0.24 Funkcja p_CalculateAvailableSeatsStudies - Michał Szymocha

Funkcja zwraca liczbę dostępnych miejsc na daanych studiach.

```
Create or alter FUNCTION p_CalculateAvailableSeatsStudies
(
    @StudiesID INT
)
returns INT
AS
BEGIN
    DECLARE @AvailableSeats INT;

    SET @AvailableSeats = (
        SELECT Studies.EnrollmentLimit - COUNT(*)
        FROM Studies JOIN StudiesDetails ON Studies.StudiesID = StudiesDetails.StudiesID
        WHERE Studies.StudiesID = @StudiesID
        GROUP BY Studies.EnrollmentLimit
    );

    RETURN @AvailableSeats;
END;
```

8 Triggery

8.0.1 Opis triggera `trg_AddPayment` - Emil Żychowicz

Cel: Trigger uruchamia się **po dodaniu płatności** do tabeli `dbo.Payments`. Automatycznie zapisuje użytkownika na odpowiednią usługę, weryfikuje daty płatności oraz typ usługi. W przypadku niespełnienia wymagań zgłasza błędy i przerywa operację.

Działanie:

1. **Walidacja**:
    - Sprawdza, czy usługa jest już opłacona (`IsReadyToParticipate = 1`).

- Weryfikuje, czy płatność została dokonana przed rozpoczęciem usługi lub co najmniej 3 dni przed jej startem (kursy, konwenty).

2. **Obsługa typów usług**: W zależności od `ServiceType`:

   - **ClassMeetingService**: Wywołuje procedury zapisujące użytkownika na zajęcia online lub stacjonarne.
   - **WebinarService**: Dodaje użytkownika do webinaru (`p_AddWebinarUser`).
   - **CourseService**: Zapisuje użytkownika na kurs, jeśli płatność była co najmniej 3 dni przed rozpoczęciem.
   - **StudiesService**: Zapisuje użytkownika na studia.
   - **ConventionService**: Zapisuje na konwent, jeśli płatność była co najmniej 3 dni przed startem.

3. **Błędy**:

   - "Service already started."– Płatność po rozpoczęciu usługi.
   - "Too late. Course/Convention starts in 3 days."– Zbyt późna płatność.

Efekt: Trigger automatyzuje proces zapisów i pilnuje terminowości płatności, zapewniając spójność danych w systemie.

```
CREATE OR ALTER TRIGGER trg_AddPayment
ON dbo.Payments
AFTER INSERT
AS
BEGIN
  --kursy - jesli opłacone ale pozniej niz 3dni przed: nie zapisuj
  --zjazdy - jesli opłacone ale pozniej niz 3dni przed: nie zapisuj
  --pobranie danych z inserted
  DECLARE @ServiceType NVARCHAR(50);
    DECLARE @ServiceID INT;
    DECLARE @OrderID INT;
  DECLARE @PaymentValue MONEY;
  DECLARE @UserID INT;
  DECLARE @PaymentDate DATETIME;

    SELECT TOP 1 @ServiceID = ServiceID, @OrderID = OrderID, @PaymentDate = PaymentDate,
        @PaymentValue = PaymentValue
    FROM inserted;

  IF EXISTS (
        SELECT 1 --1 bo nieistotne co tutaj
        FROM inserted
        INNER JOIN CONSUMER_BASKET AS cb
        ON inserted.ServiceID = cb.ServiceID AND inserted.OrderID = cb.OrderID
        WHERE cb.IsReadyToParticipate = 1 -- Sprawdzanie, czy juz opłacone
    )
    BEGIN


    IF @PaymentDate > (SELECT StartOfService FROM START_END_OF_SERVICES WHERE ServiceID
        = @ServiceID)
    BEGIN
      RAISERROR('Service already started.', 16,1);
    END

        --sprawdz servicetype
        SELECT @ServiceType = ServiceType
        FROM Services
        WHERE ServiceID = @ServiceID;

    --pobierz UserID
    SELECT @UserID = UserID
```

```
    FROM Orders
    WHERE OrderID = @OrderID

    DECLARE @LocalID INT;
        --wywołanie odpowiedniej procedury w zależności od ServiceType
        IF @ServiceType = 'ClassMeetingService'
        BEGIN
      SET @LocalID = (SELECT ClassMeetingID FROM ClassMeeting WHERE ServiceID =
          @ServiceID)
      DECLARE @MeetingType VARCHAR(50);
      SET @MeetingType = (SELECT MeetingType FROM ClassMeeting WHERE ClassMeetingID =
          @LocalID)
      IF  @MeetingType = 'online'
      BEGIN
        EXEC p_EnrollStudentInSyncClassMeeting @UserID, @LocalID
      END
      ELSE IF @MeetingType = 'offline'
      BEGIN
        EXEC p_EnrollStudentInAsyncClass @UserID, @LocalID
      END;
        END
        ELSE IF @ServiceType = 'WebinarService'
        BEGIN
      SET @LocalID = (SELECT WebinarID FROM Webinars WHERE ServiceID = @ServiceID)
      EXEC p_AddWebinarUser @UserID, @LocalID
        END
        ELSE IF @ServiceType = 'CourseService'
        BEGIN
      SET @LocalID = (SELECT CourseID FROM Courses WHERE ServiceID = @ServiceID)
      IF DATEDIFF(DAY, @PaymentDate, (SELECT CourseDate FROM Courses WHERE CourseID =
          @LocalID)) >= 3
      BEGIN
        EXEC p_AddCourseParticipant @UserID, @LocalID;
      END
      ELSE
      BEGIN
        RAISERROR('Too late. Course starts in 3 days.', 16,2)
      END
        END
        ELSE IF @ServiceType = 'StudiesService'
        BEGIN
      SET @LocalID = (SELECT StudiesID FROM Studies WHERE ServiceID = @ServiceID)
      EXEC p_EnrollStudentInStudies @UserID, @LocalID

        END
        ELSE IF @ServiceType = 'ConventionService'
        BEGIN
        SET @LocalID = (SELECT ConventionID FROM Convention WHERE ServiceID = @ServiceID)
      IF DATEDIFF(DAY, @PaymentDate, (SELECT StartDate FROM Convention WHERE
          ConventionID = @LocalID)) >= 3
      BEGIN
        EXEC p_EnrollStudentInConvention @UserID, @LocalID
      END
      BEGIN
        RAISERROR('Too late. Convention starts in 3 days.', 16, 3)
      END
        END
    END
  ELSE -- jesli nieoplacone w pelni jeszcze
  BEGIN
    IF (@ServiceType = 'CourseService' AND ISNULL((SELECT AlreadyPaidForService FROM
        CONSUMER_BASKET WHERE UserID = @UserID AND ServiceID = @ServiceID), 0) = 0
    AND @PaymentValue < (SELECT AdvanceValue FROM COURSE_INFO WHERE ServiceID =
        @ServiceID))  --dotyczy pierwszej płatności, gdy wartość wpłacona mniejsza od
        zaliczki
```

```
    BEGIN
      RAISERROR('Payment value is lower than advance value.', 16, 4);
    END
  END
END;
```

8.0.2 Opis triggera `trg_AddStudentDetails` - Jakub Kaliński

Cel: Trigger uruchamia się **po dodaniu nowego rekordu** do tabeli `dbo.StudiesDetails`. Automatycznie tworzy powiązane rekordy w tabelach `dbo.SubjectDetails` oraz `dbo.InternshipDetails`, inicjując szczegółowe dane dotyczące przedmiotów i stażu dla nowego studenta.

Działanie:

1. **Dodawanie szczegółów przedmiotów**:
   - Pobiera wszystkie przypisania przedmiotów do studiów z tabeli `dbo.SubjectStudiesAssignment` dla dodanej `StudiesID`.
   - Dla każdego przypisanego przedmiotu, tworzy rekord w tabeli `dbo.SubjectDetails` z domyślnymi wartościami ocen i frekwencji, jeśli taki rekord jeszcze nie istnieje dla danego `StudentID` i `SubjectID`.
2. **Dodawanie szczegółów stażu**:
   - Pobiera informacje o stażach z tabeli `dbo.Internship` powiązanej ze `StudiesID`.
   - Dla każdego stażu, tworzy rekord w tabeli `dbo.InternshipDetails` z domyślnymi wartościami oceny i frekwencji oraz ustaloną długością trwania, jeśli taki rekord jeszcze nie istnieje dla danego `StudentID` i `InternshipID`.

Efekt: Automatyzacja procesu tworzenia szczegółowych danych studenckich zapewnia, że każdy nowy student ma odpowiednie wpisy w tabelach dotyczących przedmiotów i staży, co ułatwia zarządzanie danymi akademickimi i stażowymi w systemie.

```
CREATE or ALTER TRIGGER [trg_AddStudentDetails]
ON [dbo].[StudiesDetails]
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    INSERT INTO dbo.SubjectDetails (SubjectID, StudentID, SubjectGrade, Attendance)
    SELECT
        s2sa.SubjectID,
        i.StudentID,
        0.0 AS SubjectGrade,
        0.0 AS Attendance
    FROM Inserted i
    INNER JOIN dbo.SubjectStudiesAssignment s2sa
        ON s2sa.StudiesID = i.StudiesID
    WHERE NOT EXISTS (
        SELECT 1
        FROM dbo.SubjectDetails sd
        WHERE sd.SubjectID = s2sa.SubjectID
          AND sd.StudentID = i.StudentID
    );

    INSERT INTO dbo.InternshipDetails (InternshipID, StudentID, Duration,
        InternshipGrade, InternshipAttendance)
    SELECT
        it.InternshipID,
        i.StudentID,
        14 AS Duration,
```

```
        0 AS InternshipGrade,
        0 AS InternshipAttendance
    FROM Inserted i
    INNER JOIN dbo.Internship it
        ON it.StudiesID = i.StudiesID
    WHERE NOT EXISTS (
        SELECT 1
        FROM dbo.InternshipDetails ind
        WHERE ind.InternshipID = it.InternshipID
          AND ind.StudentID = i.StudentID
    );

    SET NOCOUNT OFF;
END;
GO
```

8.0.3 Opis triggera `trg_AddStudentDetailsSubject` - Michał Szymocha

Cel: Trigger uruchamia się **po dodaniu nowego rekordu** do tabeli `dbo.SubjectDetails`. Automatycznie tworzy powiązane rekordy w tabelach `dbo.SyncClassDetails` oraz `dbo.AsyncClassDetails`, inicjując szczegółowe dane dotyczące zajęć synchronizowanych i asynchronicznych dla danego przedmiotu i studenta.

Działanie:

1. **Dodawanie szczegółów zajęć synchronizowanych**:
   - Łączy nowo dodany `SubjectID` z tabelą `dbo.ClassMeeting`, filtrując tylko te zajęcia, które są typu `StationaryClass` lub `OnlineLiveClass`.
   - Dla każdego takiego spotkania, tworzy rekord w tabeli `dbo.SyncClassDetails` z domyślną wartością frekwencji, jeśli taki rekord jeszcze nie istnieje dla danego `MeetingID` i `StudentID`.
2. **Dodawanie szczegółów zajęć asynchronicznych**:
   - Łączy nowo dodany `SubjectID` z tabelą `dbo.ClassMeeting`, filtrując tylko te zajęcia, które są typu `OfflineVideo`.
   - Dla każdego takiego spotkania, tworzy rekord w tabeli `dbo.AsyncClassDetails` z wartością `NULL` dla daty obejrzenia, jeśli taki rekord jeszcze nie istnieje dla danego `MeetingID` i `StudentID`.

Efekt: Automatyczne tworzenie szczegółów zajęć zapewnia, że każdy student ma odpowiednie wpisy dotyczące uczestnictwa w różnych typach zajęć, co ułatwia monitorowanie frekwencji i postępów w kursie.

```
CREATE OR ALTER TRIGGER [trg_AddStudentDetailsSubject]
ON [dbo].[SubjectDetails]
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    INSERT INTO dbo.SyncClassDetails (MeetingID, StudentID, Attendance)
    SELECT
        cm.ClassMeetingID,
        i.StudentID,
        0 AS Attendance
    FROM Inserted i
    Join ClassMeeting cm on i.SubjectID = cm.SubjectID
    WHERE
        cm.MeetingType IN ('StationaryClass', 'OnlineLiveClass')
        AND NOT EXISTS (
            SELECT 1
            FROM dbo.SyncClassDetails scd
            WHERE scd.MeetingID = cm.ClassMeetingID
```

```
                AND scd.StudentID = i.StudentID
        );

    INSERT INTO dbo.AsyncClassDetails (MeetingID, StudentID, ViewDate)
    SELECT
        cm.ClassMeetingID,
        i.StudentID,
        NULL AS ViewDate
    FROM Inserted i
    Join ClassMeeting cm on i.SubjectID = cm.SubjectID
    WHERE
        cm.MeetingType IN ('OfflineVideo')
        AND NOT EXISTS (
            SELECT 1
            FROM dbo.AsyncClassDetails acd
            WHERE acd.MeetingID = cm.ClassMeetingID
              AND acd.StudentID = i.StudentID
        );

    SET NOCOUNT OFF;
END;
```

8.0.4 Opis triggera `trg_DeleteUserFromStudies` - Michał Szymocha

Cel: Trigger uruchamia się **po usunięciu rekordu** z tabeli `dbo.StudiesDetails`. Automatycznie usuwa powiązane rekordy z tabel `dbo.SubjectDetails` oraz `dbo.InternshipDetails`, eliminując dane studenckie związane z danym użytkownikiem w kontekście przedmiotów i stażu.

Działanie:

1. **Usuwanie szczegółów przedmiotów**:
   - Identifikuje wszystkie `StudentID` usuniętych z tabeli `dbo.StudiesDetails`.
   - Usuwa wszystkie rekordy z tabeli `dbo.SubjectDetails`, które są powiązane z usuniętymi `StudentID`.
2. **Usuwanie szczegółów stażu**:
   - Identifikuje wszystkie `StudentID` usuniętych z tabeli `dbo.StudiesDetails`.
   - Usuwa wszystkie rekordy z tabeli `dbo.InternshipDetails`, które są powiązane z usuniętymi `StudentID`.

Efekt: Zapewnienie integralności danych poprzez automatyczne usuwanie powiązanych rekordów, gdy użytkownik jest usuwany ze studiów. To zapobiega pozostawianiu niepotrzebnych lub nieaktywnych danych w systemie.

```
CREATE OR ALTER TRIGGER [trg_DeleteUserFromStudies]
ON [dbo].[StudiesDetails]
AFTER DELETE
AS
BEGIN
    SET NOCOUNT ON;

    DELETE FROM dbo.SubjectDetails
    WHERE StudentID IN (SELECT StudentID FROM Deleted);

    DELETE FROM dbo.InternshipDetails
    WHERE StudentID IN (SELECT StudentID FROM Deleted);

    SET NOCOUNT OFF;
END;
```

8.0.5 Opis triggera `trg_DeleteUserFromSubject` - Jakub Kaliński

Cel: Trigger uruchamia się **po usunięciu rekordu** z tabeli `dbo.SubjectDetails`. Automatycznie usuwa powiązane rekordy z tabel `dbo.SyncClassDetails` oraz `dbo.AsyncClassDetails`, eliminując dane dotyczące uczestnictwa studenta w zajęciach synchronizowanych i asynchronicznych związanych z danym przedmiotem.

Działanie:

1. **Usuwanie szczegółów zajęć synchronizowanych**:
   - Identifikuje wszystkie `StudentID` usuniętych z tabeli `dbo.SubjectDetails`.
   - Usuwa wszystkie rekordy z tabeli `dbo.SyncClassDetails`, które są powiązane z usuniętymi `StudentID` oraz odpowiadającymi `ClassMeetingID` typu `StationaryClass` lub `OnlineLiveClass`.
2. **Usuwanie szczegółów zajęć asynchronicznych**:
   - Identifikuje wszystkie `StudentID` usuniętych z tabeli `dbo.SubjectDetails`.
   - Usuwa wszystkie rekordy z tabeli `dbo.AsyncClassDetails`, które są powiązane z usuniętymi `StudentID` oraz odpowiadającymi `ClassMeetingID` typu `OfflineVideo`.

Efekt: Utrzymanie spójności danych poprzez automatyczne usuwanie powiązanych rekordów dotyczących uczestnictwa w zajęciach, gdy student jest usuwany z konkretnego przedmiotu. To pomaga w zarządzaniu frekwencją i innymi danymi związanymi z zajęciami w systemie.

```
CREATE OR ALTER TRIGGER [trg_DeleteUserFromSubject]
ON [dbo].[SubjectDetails]
AFTER DELETE
AS
BEGIN
    SET NOCOUNT ON;

    DELETE FROM dbo.SyncClassDetails
    WHERE StudentID IN (SELECT StudentID FROM Deleted) and MeetingID in (SELECT
        ClassMeetingID FROM ClassMeeting WHERE MeetingType IN ('StationaryClass',
        'OnlineLiveClass'));

    DELETE FROM dbo.AsyncClassDetails
    WHERE StudentID IN (SELECT StudentID FROM Deleted) and MeetingID in (SELECT
        ClassMeetingID FROM ClassMeeting WHERE MeetingType IN ('OfflineVideo'));

    SET NOCOUNT OFF;
END;
```

9 Role

Każdy z członków grupy projektowej dodał swoje procedury.

9.0.1 Rola: `administrator`

Uprawnienia:

- `grant all privileges on u_szymocha.dbo to admin`

9.0.2 Rola: `director`

Uprawnienia:

- `GRANT EXECUTE ON PROCEDURE p_UpdatePrincipalAgreement to director;`

9.0.3 Rola: deans_office

Uprawnienia:

- GRANT EXECUTE ON PROCEDURE p_EnrollStudentInStudies TO deans_office;
- GRANT EXECUTE ON PROCEDURE p_DeleteSubject TO deans_office;
- GRANT EXECUTE ON PROCEDURE p_EnrollStudentInConvention TO deans_office;
- GRANT EXECUTE ON PROCEDURE p_EnrollStudentInSyncClassMeeting TO deans_office;
- GRANT EXECUTE ON PROCEDURE p_EnrollStudentInAsyncClass TO deans_office;
- GRANT EXECUTE ON PROCEDURE p_EnrollStudentInSubject TO deans_office;
- GRANT EXECUTE ON PROCEDURE p_AddService TO deans_office;
- GRANT EXECUTE ON PROCEDURE p_AddStationaryMeeting TO deans_office;
- GRANT EXECUTE ON PROCEDURE p_AddStudiesService TO deans_office;
- GRANT EXECUTE ON PROCEDURE p_AddWebinarService TO deans_office;
- GRANT EXECUTE ON PROCEDURE p_DeleteClassMeetingService TO deans_office;
- GRANT EXECUTE ON PROCEDURE p_DeleteConventionService TO deans_office;
- GRANT EXECUTE ON PROCEDURE p_DeleteCourse TO deans_office;
- GRANT EXECUTE ON PROCEDURE p_DeleteCourseService TO deans_office;
- GRANT EXECUTE ON PROCEDURE p_DeleteOfflineVideoDetails TO deans_office;
- GRANT EXECUTE ON PROCEDURE p_DeleteOnlineLiveMeetingDetails TO deans_office;
- GRANT EXECUTE ON PROCEDURE p_DeleteService TO deans_office;
- GRANT EXECUTE ON PROCEDURE p_DeleteStudiesService TO deans_office;
- GRANT EXECUTE ON PROCEDURE p_DeleteWebinarService TO deans_office;
- GRANT EXECUTE ON PROCEDURE p_EditPayment TO deans_office;
- GRANT SELECT ON VIEW CONSUMER_BASKET TO deans_office;
- GRANT SELECT ON VIEW V_StudentCollidingEvents TO deans_office;
- GRANT SELECT ON VIEW V_WebinarsWithAttendance TO deans_office;
- GRANT EXECUTE ON FUNCTION f_CalculateOrderValue TO deans_office;
- GRANT EXECUTE ON FUNCTION f_CalculatePaidOrderValue TO deans_office;
- GRANT EXECUTE ON FUNCTION f_CalculatePaidServiceValue TO deans_office;

9.0.4 Rola: coordinator_studies

Uprawnienia:

- GRANT EXECUTE ON PROCEDURE p_AddSubject TO coordinator_studies;
- GRANT EXECUTE ON PROCEDURE p_ChangeSubjectCoordinator TO coordinator_studies;
- GRANT EXECUTE ON PROCEDURE p_EditStudies TO coordinator_studies;
- GRANT EXECUTE ON PROCEDURE p_EditSubject TO coordinator_studies;
- GRANT EXECUTE ON PROCEDURE p_AddSubjectToStudies TO coordinator_studies;
- GRANT EXECUTE ON PROCEDURE p_DeleteSubjectFromStudies TO coordinator_studies;
- GRANT EXECUTE ON PROCEDURE p_InitiateInternship TO coordinator_studies;
- GRANT EXECUTE ON PROCEDURE p_CreateCourse TO coordinator_studies;
- GRANT EXECUTE ON PROCEDURE p_CreateModule TO coordinator_studies;
- GRANT EXECUTE ON PROCEDURE p_DeleteCourseParticipant TO coordinator_studies;
- GRANT EXECUTE ON PROCEDURE p_EditConventionService TO coordinator_studies;
- GRANT EXECUTE ON PROCEDURE p_EditCourses TO coordinator_studies;
- GRANT EXECUTE ON PROCEDURE p_EditCourseService TO coordinator_studies;
- GRANT EXECUTE ON PROCEDURE p_EditStudiesService TO coordinator_studies;
- GRANT SELECT ON VIEW CONVENTION_INCOME TO coordinator_studies;
- GRANT SELECT ON VIEW COURSE_INCOME TO coordinator_studies;
- GRANT SELECT ON VIEW FINANCIAL_REPORT TO coordinator_studies;
- GRANT SELECT ON VIEW FULL_PRICE TO coordinator_studies;
- GRANT SELECT ON VIEW IN_DEBT_USERS TO coordinator_studies;
- GRANT SELECT ON VIEW PARTICIPANTS_MEETINGS_FUTURE_COURSES TO coordinator_studies;
- GRANT SELECT ON VIEW NUM_OF_PARTICIPANTS_FUTURE_COURSES TO coordinator_studies;
- GRANT SELECT ON VIEW NUM_OF_PARTICIPANTS_MEETINGS_COURSES TO coordinator_studies;
- GRANT SELECT ON VIEW SERVICE_ID_INCOME TO coordinator_studies;
- GRANT SELECT ON VIEW SERVICE_USERS TO coordinator_studies;
- GRANT SELECT ON VIEW STUDIES_INCOME TO coordinator_studies;
- GRANT SELECT ON VIEW STUDIES_USERS TO coordinator_studies;

- GRANT SELECT ON VIEW WEBINAR_USERS TO coordinator_studies;
- GRANT SELECT ON VIEW WEBINAR_INCOME TO coordinator_studies;
- GRANT SELECT ON VIEW COURSE_PASSING_STATUS TO coordinator_studies;
- GRANT SELECT ON VIEW CLASS_MEETINGS_INCOME TO coordinator_studies;
- GRANT SELECT ON VIEW ATTENDANCE_RAPORT TO coordinator_studies;
- GRANT SELECT ON VIEW V_StudiesEnrollment TO coordinator_studies;
- GRANT SELECT ON VIEW V_EmployeeHierarchy TO coordinator_studies;
- GRANT SELECT ON VIEW V_EmployeeSchedule TO coordinator_studies;
- GRANT SELECT ON VIEW V_EmployeeWorkload TO coordinator_studies;
- GRANT EXECUTE ON FUNCTION f_CalculateMINRoomCapacityCourse TO coordinator_studies;
- GRANT EXECUTE ON FUNCTION f_CheckIfCourseIsPassed TO coordinator_studies;
- GRANT EXECUTE ON FUNCTION f_GetServiceValue TO coordinator_studies;
- GRANT EXECUTE ON FUNCTION f_IsReadyToParticipate TO coordinator_studies;
- GRANT EXECUTE ON FUNCTION CalculateAvailableSeatsStudies TO coordinator_studies;
- GRANT EXECUTE ON FUNCTION p_CalculateStudiesAttendance TO coordinator_studies;
- GRANT EXECUTE ON FUNCTION p_CalculateAverageNumberOfPeopleInClass TO coordinator_studies;
- GRANT EXECUTE ON FUNCTION p_CalculateMINRoomCapacity TO coordinator_studies;
- GRANT EXECUTE ON FUNCTION totalTimeSpentInClass TO coordinator_studies;
- GRANT EXECUTE ON FUNCTION p_CalculateAvailableSeatsStudies TO coordinator_studies;

9.0.5 Rola: coordinator_subject_module

Uprawnienia:

- GRANT EXECUTE ON PROCEDURE p_ChangeSubjectCoordinator TO coordinator_subject_module;
- GRANT EXECUTE ON PROCEDURE p_DeleteConvention TO coordinator_subject_module;
- GRANT EXECUTE ON PROCEDURE p_DeleteModule TO coordinator_subject_module;
- GRANT EXECUTE ON PROCEDURE p_EditClassMeetingService TO coordinator_subject_module;
- GRANT EXECUTE ON PROCEDURE p_EditModules TO coordinator_subject_module;
- GRANT SELECT ON VIEW V_ClassAttendanceAggregate TO coordinator_subject_module;
- GRANT EXECUTE ON FUNCTION f_CalculateAttendancePercentageOnModule TO coordinator_subject_module;
- GRANT EXECUTE ON FUNCTION p_CalculateSubjectAttendance TO coordinator_subject_module;

9.0.6 Rola: lecturer

Uprawnienia:

- GRANT EXECUTE ON PROCEDURE p_AddRoom TO lecturer;
- GRANT EXECUTE ON PROCEDURE p_ReserveRoom TO lecturer;
- GRANT EXECUTE ON PROCEDURE p_EditReservation TO lecturer;
- GRANT EXECUTE ON PROCEDURE p_CreateStudies TO lecturer;
- GRANT EXECUTE ON PROCEDURE p_CreateStationaryClass TO lecturer;
- GRANT EXECUTE ON PROCEDURE p_CreateOfflineVideoClass TO lecturer;
- GRANT EXECUTE ON PROCEDURE p_ChangeStudiesCoordinator TO lecturer;
- GRANT EXECUTE ON PROCEDURE p_AddConvention TO lecturer;
- GRANT EXECUTE ON PROCEDURE p_ChangeClassMeetingTeacher TO lecturer;
- GRANT EXECUTE ON PROCEDURE p_ChangeClassMeetingTranslator TO lecturer;
- GRANT EXECUTE ON PROCEDURE p_ChangeClassMeetingLanguage TO lecturer;
- GRANT EXECUTE ON PROCEDURE p_EditClassMeeting TO lecturer;
- GRANT EXECUTE ON PROCEDURE p_EditStationaryClass TO lecturer;
- GRANT EXECUTE ON PROCEDURE p_EditOnlineLiveClass TO lecturer;
- GRANT EXECUTE ON PROCEDURE p_DeleteUserClassMeetingDetails TO lecturer;
- GRANT EXECUTE ON PROCEDURE p_DeleteClassMeetingDetails TO lecturer;
- GRANT EXECUTE ON PROCEDURE p_DeleteClassMeeting TO lecturer;
- GRANT EXECUTE ON PROCEDURE p_CreateGrade TO lecturer;
- GRANT EXECUTE ON PROCEDURE p_EditGrade TO lecturer;
- GRANT EXECUTE ON PROCEDURE p_DeleteGrade TO lecturer;
- GRANT EXECUTE ON PROCEDURE p_EditOrder TO lecturer;
- GRANT EXECUTE ON PROCEDURE p_AddOrder TO lecturer;
- GRANT EXECUTE ON PROCEDURE p_AddClassMeetingService TO lecturer;
- GRANT EXECUTE ON PROCEDURE p_AddConventionService TO lecturer;

- GRANT EXECUTE ON PROCEDURE p_AddCourseService TO lecturer;
- GRANT EXECUTE ON PROCEDURE p_AddOfflineVideo TO lecturer;
- GRANT EXECUTE ON PROCEDURE p_AddOfflineVideoDetails TO lecturer;
- GRANT EXECUTE ON PROCEDURE p_AddOnlineLiveMeeting TO lecturer;
- GRANT EXECUTE ON PROCEDURE p_AddOnlineLiveMeetingDetails TO lecturer;
- GRANT EXECUTE ON PROCEDURE p_CreateOfflineVideo TO lecturer;
- GRANT EXECUTE ON PROCEDURE p_CreateStationaryMeeting TO lecturer;
- GRANT EXECUTE ON PROCEDURE p_DeleteOfflineVideo TO lecturer;
- GRANT EXECUTE ON PROCEDURE p_DeleteOnlineLiveMeeting TO lecturer;
- GRANT EXECUTE ON PROCEDURE p_DeleteStationaryMeeting TO lecturer;
- GRANT EXECUTE ON PROCEDURE p_DeleteStationaryMeetingDetails TO lecturer;
- GRANT EXECUTE ON PROCEDURE p_EditOfflineVideo TO lecturer;
- GRANT EXECUTE ON PROCEDURE p_EditOnlineLiveAttendance TO lecturer;
- GRANT EXECUTE ON PROCEDURE p_EditOnlineLiveMeeting TO lecturer;
- GRANT EXECUTE ON PROCEDURE p_EditStationaryMeeting TO lecturer;
- GRANT EXECUTE ON PROCEDURE p_EditStationaryMeetingAttendance TO lecturer;
- GRANT EXECUTE ON PROCEDURE p_EditWebinarService TO lecturer;
- GRANT EXECUTE ON PROCEDURE p_AddWebinarUser TO lecturer;
- GRANT SELECT ON VIEW ATTENDANCE_LISTS_COURSES TO lecturer;
- GRANT SELECT ON VIEW ATTENDANCE_LISTS_OFFLINEVIDEO_COURSES TO lecturer;
- GRANT SELECT ON VIEW ATTENDANCE_MEETINGS_IN_COURSES TO lecturer;
- GRANT SELECT ON VIEW CLASSMEETING_USERS TO lecturer;
- GRANT SELECT ON VIEW CONVENTION_USERS TO lecturer;
- GRANT SELECT ON VIEW COURSES_USERS TO lecturer;
- GRANT SELECT ON VIEW COURSE_SCHEDULE TO lecturer;
- GRANT SELECT ON VIEW V_StudentGrades TO lecturer;
- GRANT SELECT ON VIEW V_ConventionSchedule TO lecturer;
- GRANT SELECT ON VIEW V_SubjectMeetingSchedule TO lecturer;
- GRANT SELECT ON VIEW V_TranslatorLanguageSkill TO lecturer;
- GRANT SELECT ON VIEW V_ConventionStudents TO lecturer;
- GRANT SELECT ON VIEW V_ClassMeetingStudents TO lecturer;
- GRANT SELECT ON VIEW AllEvents TO lecturer;
- GRANT SELECT ON VIEW V_ClassAttendanceList TO lecturer;

9.0.7 Rola: student_participant

Uprawnienia:

- GRANT EXECUTE ON PROCEDURE p_AddCourseParticipant TO student_participant;
- GRANT SELECT ON VIEW START_END_OF_CLASSMEETING TO student_participant;
- GRANT SELECT ON VIEW START_END_OF_CONVENTION TO student_participant;
- GRANT SELECT ON VIEW START_END_OF_COURSES TO student_participant;
- GRANT EXECUTE ON FUNCTION f_CheckIfCourseIsPassed TO student_participant;
- GRANT SELECT ON VIEW V_SemesterSubjectsConventions TO student_participant;
- GRANT SELECT ON VIEW V_StudentsFinishedStudies TO student_participant;
- GRANT SELECT ON VIEW V_StudentSchedule TO student_participant;

9.0.8 Rola: guest

Uprawnienia:

- GRANT SELECT ON VIEW START_END_STUDIES TO guest;
- GRANT SELECT ON VIEW START_END_OF_WEBINAR TO guest;
- GRANT SELECT ON VIEW START_END_OF_COURSES TO guest;
- GRANT SELECT ON VIEW START_END_OF_CLASSMEETING TO guest;
- GRANT SELECT ON VIEW START_END_OF_CONVENTION TO guest;
- GRANT SELECT ON VIEW COURSE_INFO TO guest;
- GRANT SELECT ON VIEW V_StudiesInfo TO guest;
- GRANT SELECT ON VIEW CONSUMER_BASKET TO guest;

9.0.9 Rola: system

Uprawnienia:

- GRANT EXECUTE ON PROCEDURE p_CreateOrder TO system;
- GRANT EXECUTE ON PROCEDURE p_AddOrderDetail TO system;
- GRANT EXECUTE ON PROCEDURE p_AddPayment TO system;
- GRANT EXECUTE ON PROCEDURE p_AddStationaryMeetingDetails TO system;
- GRANT EXECUTE ON PROCEDURE p_CreateOrder TO system;
- GRANT EXECUTE ON PROCEDURE p_DeleteOrder TO system;
- GRANT EXECUTE ON PROCEDURE p_DeleteOrderDetails TO system;
- GRANT EXECUTE ON PROCEDURE p_DeletePayment TO system;
- GRANT EXECUTE ON PROCEDURE p_EditOfflineVideoDateOfViewing TO system;
- GRANT EXECUTE ON PROCEDURE p_FinalizeOrder TO system;

9.0.10 Rola: coordinator_practices

Uprawnienia:

- GRANT EXECUTE ON PROCEDURE p_CreateInternship TO coordinator_practices;
- GRANT EXECUTE ON PROCEDURE p_EditInternship TO coordinator_practices;
- GRANT EXECUTE ON PROCEDURE p_AddStudentInternship TO coordinator_practices;
- GRANT EXECUTE ON PROCEDURE p_DeleteInternship TO coordinator_practices;
- GRANT EXECUTE ON PROCEDURE p_DeleteInternshipDetails TO coordinator_practices;
- GRANT EXECUTE ON FUNCTION p_CalculateInternshipCompletion TO coordinator_practices;

10 Indeksy

10.1 ServiceUserDetails - Jakub Kaliński

10.1.1 Index: IX_ServiceUserDetails_DateOfRegistration

```
CREATE NONCLUSTERED INDEX IX_ServiceUserDetails_DateOfRegistration
    ON dbo.ServiceUserDetails (DateOfRegistration);
GO
```

10.2 Users - Jakub Kaliński

10.2.1 Index: IX_Users_DateOfBirth

```
CREATE NONCLUSTERED INDEX IX_Users_DateOfBirth
    ON dbo.Users (DateOfBirth);
GO
```

10.3 UserContact - Jakub Kaliński

10.3.1 Index: IX_UserContact_Email

```
CREATE NONCLUSTERED INDEX IX_UserContact_Email
    ON dbo.UserContact (Email);
GO
```

10.4 UserAddressDetails - Jakub Kaliński

10.4.1 Index: IX_UserAddressDetails_PostalCode

```
CREATE NONCLUSTERED INDEX IX_UserAddressDetails_PostalCode
    ON dbo.UserAddressDetails (PostalCode);
```

10.4.2 Index: IX_UserAddressDetails_LocationID

```
CREATE NONCLUSTERED INDEX IX_UserAddressDetails_LocationID
    ON dbo.UserAddressDetails (LocationID);
GO
```

10.5 Studies - Michał Szymocha

10.5.1 Index: IX_Studies_EnrollmentDeadline

```
CREATE NONCLUSTERED INDEX IX_Studies_EnrollmentDeadline
    ON dbo.Studies (EnrollmentDeadline);
GO
```

10.6 Subject - Michał Szymocha

10.6.1 Index: IX_Subject_SubjectCoordinatorID

```
CREATE NONCLUSTERED INDEX IX_Subject_SubjectCoordinatorID
    ON dbo.Subject (SubjectCoordinatorID);
GO
```

10.7 SemesterDetails - Michał Szymocha

10.7.1 Index: IX_SemesterDetails_StudiesID

```
CREATE NONCLUSTERED INDEX IX_SemesterDetails_StudiesID
    ON dbo.SemesterDetails (StudiesID);
```

10.7.2 Index: IX_SemesterDetails_StartDate

```
CREATE NONCLUSTERED INDEX IX_SemesterDetails_StartDate
    ON dbo.SemesterDetails (StartDate);
```

10.7.3 Index: IX_SemesterDetails_EndDate

```
CREATE NONCLUSTERED INDEX IX_SemesterDetails_EndDate
    ON dbo.SemesterDetails (EndDate);
GO
```

10.8 ClassMeeting - Michał Szymocha

10.8.1 Index: IX_ClassMeeting_SubjectID

```
CREATE NONCLUSTERED INDEX IX_ClassMeeting_SubjectID
    ON dbo.ClassMeeting (SubjectID);
```

10.8.2 Index: IX_ClassMeeting_TeacherID

```
CREATE NONCLUSTERED INDEX IX_ClassMeeting_TeacherID
    ON dbo.ClassMeeting (TeacherID);
```

10.8.3 Index: IX_ClassMeeting_TranslatorID

```
CREATE NONCLUSTERED INDEX IX_ClassMeeting_TranslatorID
    ON dbo.ClassMeeting (TranslatorID);
```

10.8.4 Index: IX_ClassMeeting_LanguageID

```
CREATE NONCLUSTERED INDEX IX_ClassMeeting_LanguageID
    ON dbo.ClassMeeting (LanguageID);
GO
```

10.9 StationaryClass - Michał Szymocha

10.9.1 Index: IX_StationaryClass_StartDate

```
CREATE NONCLUSTERED INDEX IX_StationaryClass_StartDate
    ON dbo.StationaryClass (StartDate);
```

10.9.2 Index: IX_StationaryClass_RoomID

```
CREATE NONCLUSTERED INDEX IX_StationaryClass_RoomID
    ON dbo.StationaryClass (RoomID);
GO
```

10.10 StationaryMeeting - Emil Żychowicz

10.10.1 Index: IX_StationaryMeeting_Module

```
CREATE NONCLUSTERED INDEX IX_StationaryMeeting_Module
    ON dbo.StationaryMeeting (ModuleID);
```

10.10.2 Index: IX_StationaryMeeting_Teacher

```
CREATE NONCLUSTERED INDEX IX_StationaryMeeting_Teacher
    ON dbo.StationaryMeeting (TeacherID);
```

10.10.3 Index: IX_StationaryMeeting_Date

```
CREATE NONCLUSTERED INDEX IX_StationaryMeeting_Date
    ON dbo.StationaryMeeting (MeetingDate);
GO
```

10.11 StationaryMeetingDetails - Emil Żychowicz

10.11.1 Index: IX_StationaryMeetingDetails_Participant

```
CREATE NONCLUSTERED INDEX IX_StationaryMeetingDetails_Participant
    ON dbo.StationaryMeetingDetails (ParticipantID);
GO
```

10.12 OnlineLiveMeeting - Emil Żychowicz

10.12.1 Index: IX_OnlineLiveMeeting_Module

```
CREATE NONCLUSTERED INDEX IX_OnlineLiveMeeting_Module
    ON dbo.OnlineLiveMeeting (ModuleID);
```

10.12.2 Index: IX_OnlineLiveMeeting_Teacher

```
CREATE NONCLUSTERED INDEX IX_OnlineLiveMeeting_Teacher
    ON dbo.OnlineLiveMeeting (TeacherID);
```

10.12.3 Index: IX_OnlineLiveMeeting_Date

```
CREATE NONCLUSTERED INDEX IX_OnlineLiveMeeting_Date
    ON dbo.OnlineLiveMeeting (MeetingDate);
GO
```

10.13 OnlineLiveMeetingDetails - Emil Żychowicz

10.13.1 Index: IX_OnlineLiveMeetingDetails_Participant

```
CREATE NONCLUSTERED INDEX IX_OnlineLiveMeetingDetails_Participant
    ON dbo.OnlineLiveMeetingDetails (ParticipantID);
GO
```

10.14 OfflineVideo - Emil Żychowicz

10.14.1 Index: IX_OfflineVideo_Module

```
CREATE NONCLUSTERED INDEX IX_OfflineVideo_Module
    ON dbo.OfflineVideo (ModuleID);
```

10.14.2 Index: IX_OfflineVideo_Teacher

```
CREATE NONCLUSTERED INDEX IX_OfflineVideo_Teacher
    ON dbo.OfflineVideo (TeacherID);
GO
```

10.15 OfflineVideoDetails - Emil Żychowicz

10.15.1 Index: IX_OfflineVideoDetails_Participant

```
CREATE NONCLUSTERED INDEX IX_OfflineVideoDetails_Participant
    ON dbo.OfflineVideoDetails (ParticipantID);
GO
```

10.16 Courses - Emil Żychowicz

10.16.1 Index: IX_Courses_ServiceID

```
CREATE NONCLUSTERED INDEX IX_Courses_ServiceID
    ON dbo.Courses (ServiceID);
```

10.16.2 Index: IX_Courses_CourseCoordinatorID

```
CREATE NONCLUSTERED INDEX IX_Courses_CourseCoordinatorID
    ON dbo.Courses (CourseCoordinatorID);
```

10.16.3 Index: IX_Courses_CourseDate

```
CREATE NONCLUSTERED INDEX IX_Courses_CourseDate
    ON dbo.Courses (CourseDate);
```

10.16.4 Index: IX_Courses_CourseName

```
CREATE NONCLUSTERED INDEX IX_Courses_CourseName
    ON dbo.Courses (CourseName);
GO
```

10.17 Modules - Emil Żychowicz

10.17.1 Index: IX_Modules_CourseID

```
CREATE NONCLUSTERED INDEX IX_Modules_CourseID
    ON dbo.Modules (CourseID);
```

10.17.2 Index: IX_Modules_ModuleCoordinatorID

```
CREATE NONCLUSTERED INDEX IX_Modules_ModuleCoordinatorID
    ON dbo.Modules (ModuleCoordinatorID);
```

10.17.3 Index: IX_Modules_LanguageID

```
CREATE NONCLUSTERED INDEX IX_Modules_LanguageID
    ON dbo.Modules (LanguageID);
```

10.17.4 Index: IX_Modules_TranslatorID

```
CREATE NONCLUSTERED INDEX IX_Modules_TranslatorID
    ON dbo.Modules (TranslatorID);
GO
```

10.18 CourseParticipants - Emil Żychowicz

10.18.1 Index: IX_CourseParticipants_CourseID

```
CREATE NONCLUSTERED INDEX IX_CourseParticipants_CourseID
    ON dbo.CourseParticipants (CourseID);
GO
```

10.19 Payments - Emil Żychowicz

10.19.1 Index: IX_Payments_OrderID

```
CREATE NONCLUSTERED INDEX IX_Payments_OrderID
    ON dbo.Payments (OrderID);
GO
```

10.20 Orders - Emil Żychowicz

10.20.1 Index: IX_Orders_UserID

```
CREATE NONCLUSTERED INDEX IX_Orders_UserID
    ON dbo.Orders (UserID);
```

10.20.2 Index: IX_Orders_OrderDate

```
CREATE NONCLUSTERED INDEX IX_Orders_OrderDate
    ON dbo.Orders (OrderDate);
GO
```

10.21 Employees - Jakub Kaliński

10.21.1 Index: IX_Employees_DateOfHire

```
CREATE NONCLUSTERED INDEX IX_Employees_DateOfHire
    ON dbo.Employees (DateOfHire);
GO
```

10.22 AsyncClassDetails - Michał Szymocha

10.22.1 Index: IX_AsyncClassDetails_StudentID

```
CREATE NONCLUSTERED INDEX IX_AsyncClassDetails_StudentID
    ON dbo.AsyncClassDetails (StudentID);
GO
```

10.23 SyncClassDetails - Michał Szymocha

10.23.1 Index: IX_SyncClassDetails_StudentID

```
CREATE NONCLUSTERED INDEX IX_SyncClassDetails_StudentID
    ON dbo.SyncClassDetails (StudentID);
GO
```

10.24 StudiesDetails - Michał Szymocha

10.24.1 Index: IX_StudiesDetails_Studies_Semester

```
CREATE NONCLUSTERED INDEX IX_StudiesDetails_Studies_Semester
    ON dbo.StudiesDetails (StudiesID, SemesterID);
GO
```

10.25 Internship - Michał Szymocha

10.25.1 Index: IX_Internship_StudiesID

```
CREATE NONCLUSTERED INDEX IX_Internship_StudiesID
    ON dbo.Internship (StudiesID);
```

10.25.2 Index: IX_Internship_StartDate

```
CREATE NONCLUSTERED INDEX IX_Internship_StartDate
    ON dbo.Internship (StartDate);
GO
```

10.26 InternshipDetails - Michał Szymocha

10.26.1 Index: IX_InternshipDetails_StudentID

```
CREATE NONCLUSTERED INDEX IX_InternshipDetails_StudentID
    ON dbo.InternshipDetails (StudentID);
GO
```

10.27 Convention - Michał Szymocha

10.27.1 Index: IX_Convention_SemesterID

```
CREATE NONCLUSTERED INDEX IX_Convention_SemesterID
    ON dbo.Convention (SemesterID);
```

10.27.2 Index: IX_Convention_StartDate

```
CREATE NONCLUSTERED INDEX IX_Convention_StartDate
    ON dbo.Convention (StartDate);
GO
```

10.28 OfflineVideoClass - Michał Szymocha

10.28.1 Index: IX_OfflineVideoClass_StartDate

```
CREATE NONCLUSTERED INDEX IX_OfflineVideoClass_StartDate
    ON dbo.OfflineVideoClass (StartDate);
```

10.28.2 Index: IX_OfflineVideoClass_Deadline

```
CREATE NONCLUSTERED INDEX IX_OfflineVideoClass_Deadline
    ON dbo.OfflineVideoClass (Deadline);
GO
```

10.29 OnlineLiveClass - Michał Szymocha

10.29.1 Index: IX_OnlineLiveClass_StartDate

```
CREATE NONCLUSTERED INDEX IX_OnlineLiveClass_StartDate
    ON dbo.OnlineLiveClass (StartDate);
GO
```

10.30 EmployeesSuperior - Jakub Kaliński

10.30.1 Index: IX_EmployeesSuperior_ReportsTo

```
CREATE NONCLUSTERED INDEX IX_EmployeesSuperior_ReportsTo
    ON dbo.EmployeesSuperior (ReportsTo);
GO
```

10.31 UserTypePermissionsHierarchy - Jakub Kaliński

10.31.1 Index: IX_UserTypePermissionsHierarchy_DirectSupervisor

```
CREATE NONCLUSTERED INDEX IX_UserTypePermissionsHierarchy_DirectSupervisor
    ON dbo.UserTypePermissionsHierarchy (DirectTypeSupervisor);
GO
```

10.32 Webinars - Jakub Kaliński - Jakub Kaliński

10.32.1 Index: IX_Webinars_TeacherID

```
CREATE NONCLUSTERED INDEX IX_Webinars_TeacherID
    ON dbo.Webinars (TeacherID);
```

10.32.2 Index: IX_Webinars_TranslatorID

```
CREATE NONCLUSTERED INDEX IX_Webinars_TranslatorID
    ON dbo.Webinars (TranslatorID);
```

10.32.3 Index: IX_Webinars_LanguageID

```
CREATE NONCLUSTERED INDEX IX_Webinars_LanguageID
    ON dbo.Webinars (LanguageID);
GO
```

10.33 WebinarDetails - Jakub Kaliński

10.33.1 Index: IX_WebinarDetails_UserID

```
CREATE NONCLUSTERED INDEX IX_WebinarDetails_UserID
    ON dbo.WebinarDetails (UserID);
GO
```

10.34 Poprawa wydajności uzyskana za pomocą indeksów

Nazwa indeksu	Bez indeksu	Z indeksem	Różnica
IX_SyncClassDetails_StudentID	45	2	–43
IX_ClassMeeting_SubjectID	20	2	–18
IX_ClassMeeting_TeacherID	20	2	–18
IX_UserAddressDetails_PostalCode	12	2	–10
IX_UserAddressDetails_LocationID	12	2	–10
IX_Payments_OrderID	12	2	–10
IX_AsyncClassDetails_StudentID	11	2	–9
IX_Webinars_TeacherID	11	2	–9
IX_Webinars_TranslatorID	11	2	–9
IX_OfflineVideoDetails_Participant	7	2	–5
IX_Orders_UserID	7	2	–5
IX_OnlineLiveMeetingDetails_Participant	5	2	–3
IX_CourseParticipants_CourseID	5	2	–3
IX_ServiceUserDetails_DateOfRegistration	5	4	–1

Tabela 1: Przykładowe dane o różnicy w liczbie odczytów logicznych przed dodaniem indeksów i po

10.35 Statystyki fragmentacji dla indeksów

Tabela 2: Statystyki fragmentacji indeksów

Nazwa tabeli	Nazwa indeksu	Fragmentacja (%)
StationaryClass	IX_StationaryClass_StartDate	50.00
Orders	IX_Orders_UserID	50.00
Orders	IX_Orders_OrderDate	50.00
Convention	Convention_pk	50.00
StationaryMeeting	StationaryMeeting_pk	50.00
StationaryMeetingDetails	IX_StationaryMeetingDetails_Participant	50.00
OfflineVideoClass	IX_OfflineVideoClass_StartDate	50.00
OfflineVideoClass	IX_OfflineVideoClass_Deadline	50.00
OnlineLiveMeetingDetails	IX_OnlineLiveMeetingDetails_Participant	50.00

Nazwa tabeli	Nazwa Indeksu	Fragmentacja (%)
CourseParticipants	CourseDetails_pk	50.00
CourseParticipants	IX_CourseParticipants_CourseID	50.00
OnlineLiveClass	IX_OnlineLiveClass_StartDate	50.00
Users	IX_Users_DateOfBirth	33.33
UserAddressDetails	IX_UserAddressDetails_LocationID	33.33
RoomDetails	RoomDetails_pk	33.33
OnlineLiveMeetingDetails	OnlineLiveMeetingDetails_pk	33.33
OfflineVideo	OfflineVideo_pk	33.33
StationaryMeetingDetails	StationaryMeetingDetails_pk	33.33
OnlineLiveMeeting	OnlineLiveMeeting_pk	33.33
ServiceUserDetails	ServiceUserDetails_pk	33.33
ServiceUserDetails	IX_ServiceUserDetails_DateOfRegistration	33.33
Payments	Payment_pk	30.00
SubjectDetails	SubjectDetails_pk	29.63
Services	Services_pk	25.00
ClassMeeting	IX_ClassMeeting_SubjectID	25.00
ClassMeeting	IX_ClassMeeting_TeacherID	25.00
Payments	IX_Payments_OrderID	25.00
StationaryClass	StationaryClass_pk	25.00
OrderDetails	OrderDetails_pk	25.00
OfflineVideoDetails	IX_OfflineVideoDetails_Participant	25.00
OfflineVideoClass	OfflineVideoClass_pk	25.00
UserAddressDetails	IX_UserAddressDetails_PostalCode	25.00
Webinars	Webinars_pk	22.22
ClassMeeting	ClassMeeting_pk	22.22
Orders	Orders_pk	20.00
OfflineVideoDetails	OfflineVideoDetails_pk	20.00
OnlineLiveClass	OnlineLiveClass_pk	20.00
UserContact	EmailUnique_uk	14.29
UserContact	IX_UserContact_Email	14.29
AsyncClassDetails	IX_AsyncClassDetails_StudentID	14.29
ClassMeetingService	ClassMeetingService_pk	12.50
WebinarDetails	WebinarDetails_pk	12.50
Users	Users_pk	12.50
AsyncClassDetails	AsyncClassDetails_pk	11.11
UserAddressDetails	UsersAddressDetails_pk	10.00
UserContact	UsersContacts_pk	9.09
SyncClassDetails	IX_SyncClassDetails_StudentID	3.03
SyncClassDetails	SyncClassDetails_pk	0.00
OnlineLiveMeeting	IX_OnlineLiveMeeting_Module	0.00
OnlineLiveMeeting	IX_OnlineLiveMeeting_Teacher	0.00
OnlineLiveMeeting	IX_OnlineLiveMeeting_Date	0.00
OfflineVideo	IX_OfflineVideo_Module	0.00
OfflineVideo	IX_OfflineVideo_Teacher	0.00
StationaryMeeting	IX_StationaryMeeting_Module	0.00
StationaryMeeting	IX_StationaryMeeting_Teacher	0.00
StationaryMeeting	IX_StationaryMeeting_Date	0.00
Webinars	IX_Webinars_TeacherID	0.00
Webinars	IX_Webinars_TranslatorID	0.00
Webinars	IX_Webinars_LanguageID	0.00
InternshipDetails	InternshipDetails_pk	0.00
InternshipDetails	IX_InternshipDetails_StudentID	0.00
Internship	Internship_pk	0.00
Internship	IX_Internship_StudiesID	0.00
Internship	IX_Internship_StartDate	0.00
Grades	Grades_pk	0.00
Studies	Studies_pk	0.00
Studies	IX_Studies_EnrollmentDeadline	0.00

Nazwa tabeli	Nazwa Indeksu	Fragmentacja (%)
Subject	Subject_pk	0.00
Subject	IX_Subject_SubjectCoordinatorID	0.00
UserTypePermissionsHierarchy	UserTypePermissionsHierarchy_pk	0.00
UserTypePermissionsHierarchy	IX_UserTypePermissionsHierarchy_DirectSupervisor	0.00
Convention	IX_Convention_SemesterID	0.00
Convention	IX_Convention_StartDate	0.00
RoomScheduleOnDate	RoomSchedule_pk	0.00
WebinarService	WebinarService_pk	0.00
Courses	CourseID	0.00
Courses	CourseID	0.00
Courses	IX_Courses_ServiceID	0.00
Courses	IX_Courses_CourseCoordinatorID	0.00
Courses	IX_Courses_CourseDate	0.00
Courses	IX_Courses_CourseName	0.00
Modules	Modules_pk	0.00
Modules	IX_Modules_CourseID	0.00
Modules	IX_Modules_ModuleCoordinatorID	0.00
Modules	IX_Modules_LanguageID	0.00
Modules	IX_Modules_TranslatorID	0.00
ConventionService	ConventionService_pk	0.00
CourseService	CourseService_pk	0.00
StudiesService	StudiesService_pk	0.00
StudiesDetails	StudiesDetails_pk	0.00
StudiesDetails	IX_StudiesDetails_Studies_Semester	0.00
SemesterDetails	SemesterDetails_pk	0.00
SemesterDetails	IX_SemesterDetails_StudiesID	0.00
SemesterDetails	IX_SemesterDetails_StartDate	0.00
SemesterDetails	IX_SemesterDetails_EndDate	0.00
SubjectStudiesAssignment	SubjectStudiesAssignment_pk	0.00
UserType	EmployeeTypes_pk	0.00
Semester	Semester_pk	0.00
Locations	Country_pk	0.00
Degrees	Degrees_pk	0.00
EmployeeDegree	DegreeDetails_pk	0.00
Employees	Employees_pk	0.00
Employees	IX_Employees_DateOfHire	0.00
EmployeesSuperior	EmployeesSuperior_pk	0.00
EmployeesSuperior	IX_EmployeesSuperior_ReportsTo	0.00
Languages	Languages_pk	0.00
Translators	Translators_pk	0.00
TranslatorsLanguages	TranslatorsLanguages_pk	0.00
Rooms	Rooms_pk	0.00