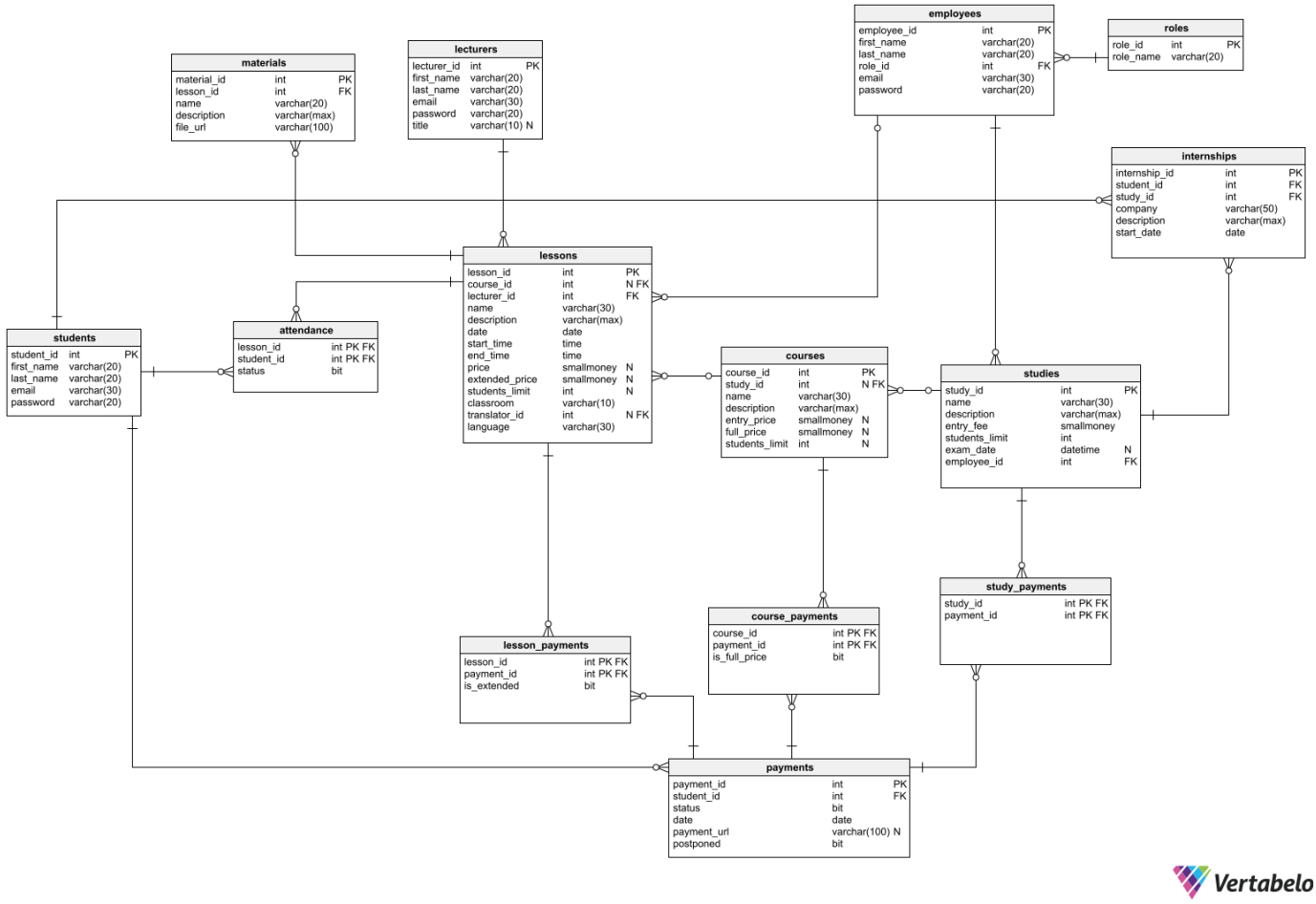
Bazy danych - System zarządzania szkoleniami

Krzysztof Śliwiński Hubert Kasprzycki Artur Dwornik

Funkcje realizowane przez system:

1. Pracownicy zarządzający wydarzeniami:
    - Zarządzanie kursami, webinarami i studiami:
        - Dodawanie nowych kursów, webinarów i studiów.
        - Edycja istniejących wydarzeń (w tym harmonogramu).
        - Usuwanie wydarzeń (zarówno bieżących, jak i archiwizowanych).
    - Zarządzanie użytkownikami:
        - Dodawanie nowych użytkowników do systemu.
        - Edycja danych użytkowników.
    - Generowanie raportów:
        - Raport ogólny z zapisanych osób na przyszłe wydarzenia.
        - Raport frekwencji na zakończonych wydarzeniach.
        - Lista obecności dla każdego szkolenia.
        - Raport bilokacji – lista osób zapisanych na kolidujące ze sobą wydarzenia.
2. Pracownicy biurowi:
    - Generowanie raportów:
        - Raport finansowy - zestawienie przychodów z poszczególnych wydarzeń.
        - Lista dłużników - sprawozdanie zalegających z płatnościami użytkowników.
        - Raport ogólny z zapisanych osób na przyszłe wydarzenia.
        - Raport frekwencji na zakończonych wydarzeniach.
        - Lista obecności dla każdego szkolenia.
        - Raport bilokacji – lista osób zapisanych na kolidujące ze sobą wydarzenia.
3. Wykładowca:
    - Edycja prowadzonych wydarzeń.
    - Oznaczanie obecności:
        - Oznaczanie obecności uczestników w trakcie spotkań.
        - Modyfikowanie listy obecności (oznaczenie odrobienia zajęć w przypadku studiów)
    - Dodawanie materiałów:
        - Wykładowca może dodawać materiały do kursów, takie jak prezentacje, pliki do pobrania itp.
4. Uczestnik:
    - Przeglądanie oferty:
        - Przeglądanie dostępnych kursów, webinarów i studiów.
        - Przeglądanie sylabusów studiów,
    - Rejestracja i płatności:
        - Rejestracja na wybrane wydarzenia.
        - Dodawanie wydarzeń do koszyka.
        - Generowanie linku do płatności.
        - Uiszczenie opłat za uczestnictwo w wydarzeniach.
    - Dostęp do nagrań:
        - Dostęp do nagrań z różnych wydarzeń online.
        - Przeglądanie dostępnych nagrań przez okres 30 dni.
    - Przeglądanie danych:
        - Uczestnik może przeglądać swoje płatności i historię uczestnictwa.

Diagram bazy danych:



Dodatkowe informacje do diagramu

- Wpis w tabeli `internships` oznacza, że student dostarczył zaświadczenie z firmy zewnętrznej o ukończeniu 14 dniowych praktyk ze 100% frekwencją w ramach danych studiów.

Tabele

Pracownicy

```
CREATE TABLE employees (
   employee_id int  NOT NULL IDENTITY(1, 1),
   first_name varchar(20)  NOT NULL,
   last_name varchar(20)  NOT NULL,
   role_id int  NOT NULL,
   email varchar(30)  NOT NULL,
   password varchar(20)  NOT NULL,
   CONSTRAINT employees_ak_1 UNIQUE (email),
   CONSTRAINT employees_pk PRIMARY KEY  (employee_id)
);
```

Role pracowników

```
CREATE TABLE roles (
   role_id int  NOT NULL IDENTITY(1,1),
   role_name varchar(20)  NOT NULL,
   CONSTRAINT roles_pk PRIMARY KEY  (role_id)
);
```

Studenci

```
CREATE TABLE students (
   student_id int  NOT NULL IDENTITY(1, 1),
   first_name varchar(20)  NOT NULL,
   last_name varchar(20)  NOT NULL,
   email varchar(30)  NOT NULL,
   password varchar(20)  NOT NULL,
   CONSTRAINT students_ak_1 UNIQUE (email),
   CONSTRAINT students_pk PRIMARY KEY  (student_id)
);
```

Lista obecności

```
CREATE TABLE attendance (
   lesson_id int  NOT NULL,
   student_id int  NOT NULL,
   status bit  NOT NULL DEFAULT 0,
   CONSTRAINT attendance_pk PRIMARY KEY  (lesson_id,student_id)
);
```

Materiały do lekcji online

```
CREATE TABLE materials (
   material_id int  NOT NULL IDENTITY(1, 1),
   lesson_id int  NOT NULL,
   name varchar(20)  NOT NULL,
   description varchar(max)  NOT NULL DEFAULT 'no description found',
   file_url varchar(100)  NOT NULL,
   CONSTRAINT materials_ak_1 UNIQUE (file_url),
   CONSTRAINT materials_pk PRIMARY KEY  (material_id)
);
```

Wykładowcy

```
CREATE TABLE lecturers (
   lecturer_id int  NOT NULL IDENTITY(1,1),
   first_name varchar(20)  NOT NULL,
   last_name varchar(20)  NOT NULL,
   email varchar(30)  NOT NULL,
   password varchar(20)  NOT NULL,
   title varchar(10)  NULL,
   CONSTRAINT lecturers_ak_1 UNIQUE (email),
   CONSTRAINT lecturers_pk PRIMARY KEY  (lecturer_id)
);
```

Praktyki zawodowe

```
CREATE TABLE internships (
   internship_id int  NOT NULL IDENTITY(1,1),
   student_id int  NOT NULL,
   study_id int  NOT NULL,
   company varchar(50)  NOT NULL,
   description varchar(max)  NOT NULL DEFAULT 'no description found',
   start_date date  NOT NULL,
   CONSTRAINT Internships_pk PRIMARY KEY  (internship_id)
);
```

Lekcje (z kursów i studiów) i webinary

```
CREATE TABLE lessons (
   lesson_id int  NOT NULL IDENTITY(1, 1),
   course_id int  NULL,
   lecturer_id int  NOT NULL,
   name varchar(30)  NOT NULL,
   description varchar(max)  NOT NULL DEFAULT 'no description found',
   date date  NOT NULL,
   start_time time  NOT NULL,
   end_time time  NOT NULL,
   price smallmoney  NOT NULL DEFAULT 0,
   extended_price smallmoney  NOT NULL DEFAULT 0,
   students_limit int  NULL,
   classroom varchar(10)  NOT NULL,
   translator_id int  NULL,
   language varchar(30)  NOT NULL,
   CONSTRAINT data CHECK (start_time < end_time),
   CONSTRAINT students_limit_lessons CHECK (students_limit > 0),
   CONSTRAINT price CHECK (price >= 0),
   CONSTRAINT extended_price CHECK (extended_price >= price),
   CONSTRAINT lessons_pk PRIMARY KEY  (lesson_id)
);
```

Kursy

```
CREATE TABLE courses (
   course_id int  NOT NULL IDENTITY(1, 1),
   study_id int  NULL,
   name varchar(30)  NOT NULL,
   description varchar(max)  NOT NULL DEFAULT 'no description found',
   entry_price smallmoney  NOT NULL DEFAULT 0,
   full_price smallmoney  NOT NULL DEFAULT 0,
   students_limit int  NULL,
   CONSTRAINT entry_price CHECK (entry_price >= 0),
   CONSTRAINT full_price CHECK (full_price >= 0),
   CONSTRAINT students_limit_courses CHECK (students_limit > 0),
   CONSTRAINT courses_pk PRIMARY KEY  (course_id)
);
```

Studia

```
CREATE TABLE studies (
   study_id int  NOT NULL IDENTITY(1, 1),
   name varchar(30)  NOT NULL,
   description varchar(max)  NOT NULL DEFAULT 'no description found',
   entry_fee smallmoney  NOT NULL,
   students_limit int  NOT NULL,
   exam_date datetime  NULL,
   employee_id int  NOT NULL,
   CONSTRAINT entry_fee CHECK (entry_fee > 0),
   CONSTRAINT students_limit_studies CHECK (students_limit > 0),
   CONSTRAINT studies_pk PRIMARY KEY  (study_id)
);
```

Płatności

```
CREATE TABLE payments (
    payment_id int  NOT NULL IDENTITY(1, 1),
    student_id int  NOT NULL,
    status bit  NOT NULL DEFAULT 0,
    date date  NOT NULL,
    payment_url varchar(100)  NULL,
    postponed bit  NOT NULL,
    CONSTRAINT payments_ak_1 UNIQUE (payment_url),
    CONSTRAINT payments_pk PRIMARY KEY  (payment_id)
);
```

Płatności za lekcje

```
CREATE TABLE lesson_payments (
    lesson_id int  NOT NULL,
    payment_id int  NOT NULL,
    is_extended bit  NOT NULL,
    CONSTRAINT lesson_payments_pk PRIMARY KEY  (lesson_id,payment_id)
);
```

Płatności za kursy

```
CREATE TABLE course_payments (
    course_id int  NOT NULL,
    payment_id int  NOT NULL,
    is_full_price bit  NOT NULL DEFAULT 0,
    CONSTRAINT course_payments_pk PRIMARY KEY  (course_id,payment_id)
);
```

Płatności za studia

```
CREATE TABLE study_payments (
    study_id int  NOT NULL,
    payment_id int  NOT NULL,
    CONSTRAINT study_payments_pk PRIMARY KEY  (study_id,payment_id)
);
```

Widoki

1. Lista „dłużników”

Widoki wyświetlające id, imię oraz nazwisko studentów, którzy zalegają z płatnościami za odpowiednio: studia, kursy i webinary, oraz zbiorczo za wszystkie te kategorie.

Dłużnicy dla studiów

```
CREATE VIEW studies_debtors_list AS
    SELECT s.student_id,
           s.first_name,
           s.last_name,
           SUM(IIF(sp.payment_id IS NULL, l.extended_price, l.price)) AS debt
    FROM students s
        JOIN attendance a
            ON s.student_id = a.student_id AND a.status = 1
        JOIN lessons l
            ON a.lesson_id = l.lesson_id
        JOIN courses c
            ON l.course_id = c.course_id
        JOIN studies st
            ON c.study_id = st.study_id
        LEFT JOIN payments p
            ON s.student_id = p.student_id
        LEFT JOIN lesson_payments lp
            ON p.payment_id = lp.payment_id AND a.lesson_id = lp.lesson_id
        LEFT JOIN study_payments sp
            ON p.payment_id = sp.payment_id AND st.study_id = sp.study_id
    WHERE p.payment_id IS NULL OR (p.postponed = 0 AND (lp.payment_id IS NULL OR sp.payment_id IS NULL OR p.status = 0))
    GROUP BY s.student_id, s.first_name, s.last_name
```

Dłużnicy dla kursów

```
CREATE VIEW courses_debtors_list AS
    SELECT s.student_id,
           s.first_name,
           s.last_name,
           SUM(IIF(cp.is_full_price = 0, c.full_price - c.entry_price, c.full_price)) AS debt
    FROM students s
        JOIN attendance a
            ON s.student_id = a.student_id AND a.status = 1
        JOIN lessons l
            ON a.lesson_id = l.lesson_id
        JOIN courses c
            ON l.course_id = c.course_id AND c.study_id IS NULL
        LEFT JOIN payments p
            ON s.student_id = p.student_id
        LEFT JOIN course_payments cp
            ON p.payment_id = cp.payment_id AND c.course_id = cp.course_id
    WHERE p.payment_id IS NULL OR (p.postponed = 0 AND (cp.payment_id IS NULL OR p.status = 0 OR cp.is_full_price = 0))
    GROUP BY s.student_id, s.first_name, s.last_name
```

Dłużnicy dla webinarów

```
CREATE VIEW webinars_debtors_list AS
    SELECT s.student_id,
           s.first_name,
           s.last_name,
           SUM(l.price) AS debt
    FROM students s
        JOIN attendance a
            ON s.student_id = a.student_id AND a.status = 1
        JOIN lessons l
            ON a.lesson_id = l.lesson_id AND l.course_id is null
        LEFT JOIN payments p
            ON s.student_id = p.student_id
        LEFT JOIN lesson_payments lp
            ON p.payment_id = lp.payment_id AND a.lesson_id = lp.lesson_id
    WHERE p.payment_id IS NULL OR (p.postponed = 0 AND (lp.payment_id IS NULL OR p.status = 0))
    GROUP BY s.student_id, s.first_name, s.last_name
```

Wszyscy dłużnicy

```
CREATE OR ALTER VIEW debtors_list AS
    with t as (SELECT * FROM webinars_debtors_list wdl
               UNION
               SELECT * FROM courses_debtors_list
               UNION
               SELECT * FROM studies_debtors_list)
    SELECT t.student_id, t.first_name, t.last_name, SUM(t.debt) AS debt
    FROM t
    GROUP BY t.student_id, t.first_name, t.last_name
```

Przykładowy rezultat widoku

student_id	first_name	last_name	debt
1	Patrick	Stevens	6373.5000
2	Daniel	Brown	6436.5000
3	Michael	Griffin	6279.0000
4	Miguel	Olson	6405.0000
5	Anthony	Lowe	6541.5000
6	Samantha	Schaefer	6625.5000

2. Ogólny raport dotyczący liczby zapisanych osób na wydarzenia

Widok wyświetlający id lekcji, nazwę lekcji, datę lekcji, liczbę studentów zapisanych na daną lekcję oraz formę danej lekcji (stacjonarna / zdalna) dla wszystkich lekcji.

```
CREATE VIEW students_registered_count AS
    WITH
        studiesStudentsCount AS (
            SELECT s.study_id, count(*) as "count"
            FROM studies s
            JOIN study_payments sp ON s.study_id=sp.study_id
            JOIN payments p ON sp.payment_id=p.payment_id
            WHERE p.[status]=1 OR (p.[status]=0 AND p.postponed=1)
            GROUP BY s.study_id
        ),
        coursesStudentsCount AS (
            SELECT c.course_id, count(*) as "count"
            FROM courses c
            JOIN course_payments cp ON c.course_id=cp.course_id
            JOIN payments p ON cp.payment_id=p.payment_id
            WHERE p.[status]=1 OR (p.[status]=0 AND p.postponed=1)
            GROUP BY c.course_id
        ),
        lessonsStudentsCount AS (
            SELECT l.lesson_id, count(*) as "count"
            FROM lessons l
            JOIN lesson_payments lp ON l.lesson_id=lp.lesson_id
            JOIN payments p ON lp.payment_id=p.payment_id
            WHERE p.[status]=1 OR (p.[status]=0 AND p.postponed=1)
            GROUP BY l.lesson_id
        ),
        extraStudentsCount AS (
            SELECT l.lesson_id, count(*) as "count"
            FROM lessons l
            JOIN lesson_payments lp ON l.lesson_id=lp.lesson_id
            JOIN payments p ON lp.payment_id=p.payment_id
            WHERE lp.is_extended=1 AND (p.[status]=1 OR (p.[status]=0 AND p.postponed=1))
            GROUP BY l.lesson_id
        )
    SELECT
        l.lesson_id,
        l.name,
        l.date,
        (
            CASE
                WHEN l.course_id IS NULL THEN lsc.count
                WHEN c.study_id IS NULL THEN csc.count
                ELSE ssc.count + ISNULL(esc.count,0)
            END
        ) AS "count",
        (
            CASE
                WHEN l.classroom='online' THEN 'remote'
                ELSE 'stationary'
            END
        ) as "lesson form"
    FROM lessons l
    LEFT JOIN courses c on l.course_id=c.course_id
    LEFT JOIN studies s on c.study_id=s.study_id
    LEFT JOIN lessonsStudentsCount lsc ON l.lesson_id=lsc.lesson_id
    LEFT JOIN coursesStudentsCount csc ON c.course_id=csc.course_id
    LEFT JOIN studiesStudentsCount ssc ON s.study_id=ssc.study_id
    LEFT JOIN extraStudentsCount esc ON l.lesson_id=esc.lesson_id
```

Przykładowy rezultat widoku

lesson_id	name	date	count	lesson form
1	coaching lesson	2024-01-01	35	stationary
2	maybeing lesson	2024-01-07	35	stationary
3	stageing lesson	2024-01-13	35	remote
4	catching lesson	2024-01-19	35	stationary
5	continueing lesson	2024-01-25	35	stationary
6	cameraing lesson	2024-01-31	35	stationary

3. Ogólny raport dotyczący liczby zapisanych osób na przyszłe wydarzenia

Widok wyświetlający id lekcji, nazwę lekcji, datę lekcji, liczbę zapisanych studentów na daną lekcję oraz formę danej lekcji (stacjonarna / zdalna) dla lekcji, które się jeszcze nie odbyły.

```
CREATE VIEW students_registered_future_count AS (
    SELECT src.lesson_id, src.name, src.date, src.[count], src.[lesson form]
    FROM students_registered_count src
    JOIN lessons l ON src.lesson_id=l.lesson_id
    WHERE l.[date] > GETDATE()
)
```

Przykładowy rezultat widoku

lesson_id	name	date	count	lesson form
5	continueing lesson	2024-01-25	35	stationary
6	cameraing lesson	2024-01-31	35	stationary
7	civiling lesson	2024-02-01	35	stationary
8	necessarying lesson	2024-02-06	35	stationary
9	developmenting lesson	2024-02-12	35	stationary
10	perhapsing lesson	2024-02-17	35	stationary

4. Ogólny raport dotyczący frekwencji na zakończonych już wydarzeniach

Widok wyświetlający id lekcji, nazwę lekcji, datę lekcji oraz frekwencję na danej lekcji dla wszystkich lekcji.

```
CREATE VIEW attendance_percentage_report AS
    WITH
        attendanceTotal AS (
            SELECT
                l.lesson_id,
                l.name,
                l.date,
                COUNT(*) AS count
            FROM lessons l
            JOIN attendance a on l.lesson_id=a.lesson_id
            WHERE l.date < GETDATE()
            GROUP BY l.lesson_id, l.name, l.date
        ),
        attendancePresent AS (
            SELECT
                l.lesson_id,
                COUNT(*) AS count
            FROM lessons l
            JOIN attendance a ON l.lesson_id=a.lesson_id
            WHERE l.date < GETDATE() AND a.[status]=1
            GROUP BY l.lesson_id
        )
    SELECT
        att.lesson_id,
        att.name,
        att.date,
        CAST((CAST(atp.count AS float)/CAST(att.count AS float)) AS numeric(20,2)) AS "Attendance Percentage"
    FROM attendanceTotal att
    JOIN attendancePresent atp on att.lesson_id=atp.lesson_id
```

Przykładowy rezultat widoku

lesson_id	name	date	Attendance Percentage
1	coaching lesson	2024-01-01	0.94
2	maybeing lesson	2024-01-07	0.94
3	stageing lesson	2024-01-13	0.83
4	catching lesson	2024-01-19	0.94
49	businessing lesson	2023-05-01	0.90
50	officering lesson	2023-05-04	0.86

5. Lista Obecności

Widok wyświetlający listę obecności: id lekcji, datę lekcji, imię i nazwisko studenta oraz status obecności (obecny / nieobecny), dla wszystkich lekcji.

```
CREATE VIEW attendance_list AS (
    SELECT
        l.lesson_id,
        l.[date],
        s.first_name,
        s.last_name,
        (
            CASE
                WHEN a.status=0 THEN 'ABSENT'
                ELSE 'PRESENT'
            END
        ) AS "status"
    FROM lessons l
    JOIN attendance a ON l.lesson_id=a.lesson_id
    JOIN students s ON a.student_id=s.student_id
)
```

Przykładowy rezultat widoku

lesson_id	date	first_name	last_name	status
1	2024-01-01	Patrick	Stevens	PRESENT
1	2024-01-01	Daniel	Brown	PRESENT
1	2024-01-01	Michael	Griffin	PRESENT
1	2024-01-01	Miguel	Olson	PRESENT
1	2024-01-01	Anthony	Lowe	PRESENT
1	2024-01-01	Samantha	Schaefer	PRESENT

6. Raport Bilokacji

Widok wyświetlający listę studentów, którzy mają kolidujące ze sobą lekcje: id studenta, data lekcji, id pierwszej kolidującej lekcji i jej nazwa, id drugiej kolidującej lekcji i jej nazwa.

```
CREATE VIEW bilocation_report AS
WITH
myData AS (
    SELECT s.student_id, l.lesson_id, l.name, l.[date], l.start_time, l.end_time
    FROM students s
    JOIN attendance a ON a.student_id=s.student_id
    JOIN lessons l ON l.lesson_id=a.lesson_id
)
SELECT DISTINCT md1.student_id, md1.date, md1.lesson_id as "lesson 1 id", md1.name as "lesson 1", md2.lesson_id as "lesson 2 id",
md2.name as "lesson 2"
FROM myData md1
JOIN myData md2 ON md1.student_id=md2.student_id
WHERE md1.[date]=md2.[date] AND
    (
        (md2.start_time >= md1.start_time AND md1.end_time >= md2.start_time)
    OR
        (md1.start_time >= md2.start_time AND md2.end_time >= md1.start_time)
    )
    AND md1.lesson_id<>md2.lesson_id
    AND md1.lesson_id > md2.lesson_id
```

Przykładowy rezultat widoku

student_id	date	lesson 1 id	lesson 1	lesson 2 id	lesson 2
65	2024-03-01	109	itsing lesson	99	duringing lesson
66	2024-03-01	109	itsing lesson	99	duringing lesson
67	2024-03-01	109	itsing lesson	99	duringing lesson
68	2024-03-01	109	itsing lesson	99	duringing lesson
69	2024-03-01	109	itsing lesson	99	duringing lesson
70	2024-03-01	109	itsing lesson	99	duringing lesson

Procedury

1. Wyświetl koszyk danego użytkownika (lekcje)

Procedura wyświetlająca koszyk (zawierający wyłącznie lekcje) danego studenta.

```
CREATE PROCEDURE student_cart_lessons_info(@student_id INT)
AS
BEGIN
    BEGIN TRY
        IF NOT EXISTS(
            SELECT *
            FROM students
            WHERE @student_id=student_id
        )
        BEGIN
            THROW 53000, N'There is no student with given ID', 1
        END
        SELECT
            l.lesson_id,
            l.name,
            l.[description],
            l.[date],
            l.start_time,
            l.end_time,
            CASE
                WHEN lp.is_extended=1 THEN l.extended_price
                ELSE l.price
            END AS "price",
            l.classroom,
            l.[language]
        FROM payments p
        JOIN lesson_payments lp ON p.payment_id=lp.payment_id
        JOIN lessons l ON l.lesson_id=lp.lesson_id
        WHERE p.student_id=@student_id AND p.payment_url IS NULL
    END TRY
    BEGIN CATCH
        DECLARE @msg NVARCHAR(2048) = N'ERROR: ' + ERROR_MESSAGE();
        THROW 53000, @msg, 1;
    END CATCH
END
```

Przykładowy rezultat procedury dla @student_id = 101

lesson_id	name	description	date	start_time	end_time	price	classroom	language
71	likelying lesson	likelying lesson description	2023-08-31	14:48:00.0000000	16:18:00.0000000	79.9900	C1	Spanish
72	pressureing lesson	pressureing lesson description	2023-09-30	16:40:00.0000000	18:10:00.0000000	59.9900	A2	Polish

2. Wyświetl koszyk danego użytkownika (kursy)

Procedura wyświetlająca koszyk (zawierający wyłącznie kursy) danego studenta.

```
CREATE PROCEDURE student_cart_courses_info(@student_id INT)
AS
BEGIN
    BEGIN TRY
        IF NOT EXISTS(
            SELECT *
            FROM students
            WHERE @student_id=student_id
        )
        BEGIN
            THROW 53000, N'There is no student with given ID', 1
        END
        SELECT
            c.course_id,
            c.name,
            c.[description],
            c.entry_price,
            c.full_price
        FROM payments p
        JOIN course_payments cp ON p.payment_id=cp.payment_id
        JOIN courses c ON c.course_id=cp.course_id
        WHERE p.student_id=@student_id AND p.payment_url IS NULL
    END TRY
    BEGIN CATCH
        DECLARE @msg NVARCHAR(2048) = N'ERROR: ' + ERROR_MESSAGE();
        THROW 53000, @msg, 1;
    END CATCH
END
```

Przykładowy rezultat procedury dla @student_id = 101

course_id	name	description	entry_price	full_price
15	pering course	pering course description	120.0000	1280.0000
16	ining course	ining course description	95.0000	605.0000

3. Wyświetl koszyk danego użytkownika (studia)

Procedura wyświetlająca koszyk (zawierający wyłącznie studia) danego studenta.

```
CREATE PROCEDURE student_cart_studies_info(@student_id INT)
AS
BEGIN
    BEGIN TRY
        IF NOT EXISTS(
            SELECT *
            FROM students
            WHERE @student_id=student_id
        )
        BEGIN
            THROW 53000, N'There is no student with given ID', 1
        END
        SELECT
            s.study_id,
            s.name,
            s.[description],
            s.entry_fee,
            s.exam_date
        FROM payments p
        JOIN study_payments sp ON p.payment_id=sp.payment_id
        JOIN studies s ON s.study_id=sp.study_id
        WHERE p.student_id=@student_id AND p.payment_url IS NULL
    END TRY
    BEGIN CATCH
        DECLARE @msg NVARCHAR(2048) = N'ERROR: ' + ERROR_MESSAGE();
        THROW 53000, @msg, 1;
    END CATCH
END
```

Przykładowy rezultat procedury dla @student_id = 101

study_id	name	description	entry_fee	exam_date
2	responsibilitying study	responsibilitying study description	500.0000	2024-01-25 00:00:00.000

4. Przeglądaj historię uczestnictwa

Procedura wyświetlająca historię uczestnictwa w lekcjach danego studenta.

```
CREATE PROCEDURE student_attendance_history(@student_id INT)
AS
BEGIN
    BEGIN TRY
        IF NOT EXISTS(
            SELECT *
            FROM students
            WHERE @student_id=student_id
        )
        BEGIN
            THROW 53000, N'There is no student with given ID', 1
        END
        SELECT
            l.lesson_id,
            l.name,
            l.[date]
        FROM attendance a
        JOIN lessons l ON a.lesson_id=l.lesson_id
        WHERE a.student_id=@student_id AND a.[status]=1
        ORDER BY l.[date] ASC
    END TRY
    BEGIN CATCH
        DECLARE @msg NVARCHAR(2048) = N'ERROR: ' + ERROR_MESSAGE();
        THROW 53000, @msg, 1;
    END CATCH
END
```

Przykładowy rezultat procedury dla @student_id = 50

lesson_id	name	date
74	theying lesson	2023-02-01
75	projecting lesson	2023-02-15
76	analysising lesson	2023-03-01
77	performing lesson	2023-03-15
78	parenting lesson	2023-03-29
79	itselfing lesson	2023-04-01

5. Aktualizuj obecność

Procedura pozwalająca zaktualizować status obecności danego studenta na danej lekcji.

```
-- 5. Aktualizuj obecność
CREATE PROCEDURE update_attendance(@lesson_id INT, @student_id INT, @status BIT)
AS
BEGIN
    BEGIN TRY
        IF NOT EXISTS(
            SELECT *
            FROM attendance
            WHERE lesson_id=@lesson_id
        )
        BEGIN
            THROW 53000, N'There is no lesson with given ID', 1
        END
        IF NOT EXISTS(
            SELECT *
            FROM students
            WHERE student_id=@student_id
        )
        BEGIN
            THROW 53000, N'There is no student with given ID', 1
        END
        IF NOT EXISTS(
            SELECT *
            FROM attendance
            WHERE student_id=@student_id
        )
        BEGIN
            THROW 53000, N'There is no attendance for student with given ID', 1
        END
        UPDATE attendance
        SET [status]=@status
        WHERE lesson_id=@lesson_id AND student_id=@student_id
    END TRY
    BEGIN CATCH
        DECLARE @msg NVARCHAR(2048) = N'ERROR: ' + ERROR_MESSAGE();
        THROW 53000, @msg, 1
    END CATCH
END
```

6. Dodaj lekcję w ramach studiów

Procedura dodająca lekcję w ramach oferty studiów.

```
CREATE PROCEDURE add_study_lesson(
    @lecturer_id INT,
    @course_id INT,
    @name VARCHAR(30),
    @date DATE,
    @start_time TIME,
    @end_time TIME,
    @classroom VARCHAR(10),
    @language VARCHAR(30),
    @description VARCHAR(max) = 'No description',
    @price SMALLMONEY = 0,
    @extended_price SMALLMONEY = 0,
    @students_limit INT = null,
    @translator_id INT = null
)
AS
BEGIN
    BEGIN TRY
        IF NOT exists(select * from lecturers where lecturer_id = @lecturer_id)
            BEGIN
                THROW 53000, N'Uknown lecturer!', 1;
            END

        IF NOT exists(select * from courses where course_id = @course_id and study_id is not null)
            BEGIN
                THROW 53000, N'Uknown course or course is not from any studies!', 1;
            END

        IF @language <> 'Polish' and @translator_id is null
            BEGIN
               THROW 53000, N'Lack of translator', 1;
            END

        IF @classroom = 'Online' and @students_limit is not null
            BEGIN
                THROW 53000, N'Online lessons cannot be limited', 1;
            END

        DECLARE @study_students_limit INT
        SET @study_students_limit = (select s.students_limit
                                     from studies s
                                     where s.study_id = (select c.study_id
                                                         from courses c
                                                         where c.course_id = @course_id))

        IF @students_limit is not null and @students_limit < @study_students_limit
            BEGIN
                THROW 53000, N'Incorrect students limit, limit is less than study limit', 1;
            end

        INSERT INTO lessons (lecturer_id, name, description, date, start_time, end_time, price, extended_price, students_limit,
                             classroom, translator_id, language)
        VALUES (@lecturer_id, @name, @description, @date, @start_time, @end_time, @price, @extended_price, @students_limit,
@classroom,
                @translator_id, @language)
    END TRY
    BEGIN CATCH
        DECLARE @msg VARCHAR(2048) = N'ERROR: ' + ERROR_MESSAGE();
        THROW 53000, @msg, 1;
    END CATCH
END
```

7. Dodaj lekcję z kursu

Procedura dodająca lekcję w ramach oferty kursu.

```
CREATE PROCEDURE add_course_lesson(
    @lecturer_id INT,
    @course_id INT,
    @name VARCHAR(30),
    @date DATE,
    @start_time TIME,
    @end_time TIME,
    @classroom VARCHAR(10),
    @language VARCHAR(30),
    @description VARCHAR(max) = 'No description',
    @translator_id INT = null
)
AS
BEGIN
    BEGIN TRY
        IF NOT exists(select * from lecturers where lecturer_id = @lecturer_id)
            BEGIN
                THROW 53000, N'Uknown lecturer!', 1;
            END

        IF NOT exists(select * from courses where course_id = @course_id and study_id is null)
            BEGIN
                THROW 53000, N'Uknown course', 1;
            END

        IF @language <> 'Polish' and @translator_id is null
            BEGIN
               THROW 53000, N'Lack of translator', 1;
            END

        DECLARE @course_students_limit INT
        SET @course_students_limit = (select c.students_limit
                                      from courses c
                                      where c.course_id = @course_id)

        INSERT INTO lessons (lecturer_id, name, description, date, start_time, end_time, students_limit,
                             classroom, translator_id, language)
        VALUES (@lecturer_id, @name, @description, @date, @start_time, @end_time, @course_students_limit, @classroom,
                @translator_id, @language)
    END TRY
    BEGIN CATCH
        DECLARE @msg VARCHAR(2048) = N'ERROR: ' + ERROR_MESSAGE();
        THROW 53000, @msg, 1;
    END CATCH
END
```

8. Dodaj webinar

Procedura dodająca webinar.

```
CREATE PROCEDURE add_webinar(
    @lecturer_id INT,
    @name VARCHAR(30),
    @date DATE,
    @start_time TIME,
    @end_time TIME,
    @classroom VARCHAR(10),
    @language VARCHAR(30),
    @description VARCHAR(max) = 'No description',
    @price SMALLMONEY = 0,
    @students_limit INT = null,
    @translator_id INT = null
)
AS
BEGIN
    BEGIN TRY
        IF NOT exists(select * from lecturers where lecturer_id = @lecturer_id)
            BEGIN
                THROW 53000, N'Uknown lecturer!', 1;
            END

        IF @language <> 'Polish' and @translator_id is null
            BEGIN
               THROW 53000, N'Lack of translator', 1;
            END

        IF @classroom = 'Online' and @students_limit is not null
            BEGIN
                THROW 5300, N'Online lessons cannot be limited', 1;
            END

        INSERT INTO lessons (lecturer_id, name, description, date, start_time, end_time, price, extended_price, students_limit,
                             classroom, translator_id, language)
        VALUES (@lecturer_id, @name, @description, @date, @start_time, @end_time, @price, @price, @students_limit, @classroom,
                @translator_id, @language)
    END TRY
    BEGIN CATCH
        DECLARE @msg VARCHAR(2048) = N'ERROR: ' + ERROR_MESSAGE();
        THROW 53000, @msg, 1;
    END CATCH
END
```

9. Dodawanie lekcji do koszyka

Procedura dodająca lekcję do koszyka danego studenta.

```
CREATE PROCEDURE add_lesson_to_cart(@payment_id INT, @lesson_id INT, @student_id INT)
AS
BEGIN
    BEGIN TRY
        IF NOT EXISTS(
            SELECT *
            FROM payments
            WHERE payment_id=@payment_id
        )
        BEGIN
            THROW 53000, N'There is no payment with given ID', 1
        END
        IF NOT EXISTS(
            SELECT *
            FROM students
            WHERE student_id=@student_id
        )
        BEGIN
            THROW 53000, N'There is no student with given ID', 1
        END
        IF NOT EXISTS(
            SELECT *
            FROM lessons
            WHERE lesson_id=@lesson_id
        )
        BEGIN
            THROW 53000, N'There is no lesson with given ID', 1
        END
        DECLARE @is_extended INT
        DECLARE @lesson_studies INT
        SET @is_extended = 0
        SET @lesson_studies = dbo.get_lesson_studies(@lesson_id)

        IF(@lesson_studies IS NOT NULL)
            BEGIN
                IF (
                    NOT EXISTS(
                    SELECT p.payment_id
                    FROM study_payments sp
                    JOIN payments p ON sp.payment_id=p.payment_id
                    WHERE p.student_id=@student_id
                    AND (p.[status]=1 OR p.postponed=1)
                    AND dbo.get_lesson_studies(@lesson_id)=sp.study_id
                    )
                )
                BEGIN
                    SET @is_extended=1
                END
            END

        IF NOT EXISTS
            (SELECT p.status FROM payments p WHERE p.payment_id=@payment_id AND p.[status]=0 AND p.postponed=0)
        BEGIN
            THROW 53000, N'Cannot add the item to this payment',  1
        END

        IF dbo.calc_lesson_vacancy_amount(@lesson_id) <= 0
        BEGIN
            THROW 53000, N'The are no vacancies for this lesson', 1
        END

        IF(
            (SELECT l.course_id FROM lessons l WHERE l.lesson_id=@lesson_id) IS NOT NULL
            AND @lesson_studies IS NULL
            )
        BEGIN
            THROW 53000, N'This lesson is not available', 1
        END

        INSERT INTO lesson_payments(lesson_id, payment_id, is_extended)
        VALUES (@lesson_id, @payment_id, @is_extended)
    END TRY
    BEGIN CATCH
        DECLARE @msg NVARCHAR(2048) = N'ERROR: ' + ERROR_MESSAGE();
        THROW 53000, @msg, 1
    END CATCH
END
```

10. Dodawanie kursu do koszyka

Procedura dodająca kurs do koszyka danego studenta.

```
CREATE PROCEDURE add_course_to_cart(@payment_id INT, @course_id INT, @student_id INT, @is_full_price INT)
AS
BEGIN
    BEGIN TRY
        IF NOT EXISTS(
            SELECT *
            FROM payments
            WHERE payment_id=@payment_id
        )
        BEGIN
            THROW 53000, N'There is no payment with given ID', 1
        END
        IF NOT EXISTS(
            SELECT *
            FROM students
            WHERE student_id=@student_id
        )
        BEGIN
            THROW 53000, N'There is no student with given ID', 1
        END
        IF NOT EXISTS(
            SELECT *
            FROM courses
            WHERE course_id=@course_id
        )
        BEGIN
            THROW 53000, N'There is no course with given ID', 1
        END

        DECLARE @course_studies INT
        SET @course_studies = (SELECT c.study_id FROM courses c WHERE c.course_id=@course_id)

        IF NOT EXISTS
        (SELECT p.status FROM payments p WHERE p.payment_id=@payment_id AND p.[status]=0 AND p.postponed=0)
        BEGIN
           THROW 53000, N'Cannot add the item to this payment',  1
        END

        IF dbo.calc_course_vacancy_amount(@course_id) <= 0
        BEGIN
            THROW 53000, N'The are no vacancies for this course', 1
        END

        IF @course_studies IS NOT NULL
        BEGIN
            THROW 53000, N'This course is not available', 1
        END

        INSERT INTO course_payments(course_id, payment_id, is_full_price)
        VALUES (@course_id, @payment_id, @is_full_price)
    END TRY
    BEGIN CATCH
        DECLARE @msg NVARCHAR(2048) = N'ERROR: ' + ERROR_MESSAGE();
        THROW 53000, @msg, 1
    END CATCH
END
```

11. Dodawanie studiów do koszyka

Procedura dodająca studia do koszyka danego studenta.

```
CREATE PROCEDURE add_study_to_cart(@payment_id INT, @study_id INT, @student_id INT)
AS
BEGIN
    BEGIN TRY
        IF NOT EXISTS(
            SELECT *
            FROM payments
            WHERE payment_id=@payment_id
        )
        BEGIN
            THROW 53000, N'There is no payment with given ID', 1
        END
        IF NOT EXISTS(
            SELECT *
            FROM students
            WHERE student_id=@student_id
        )
        BEGIN
            THROW 53000, N'There is no student with given ID', 1
        END
        IF NOT EXISTS(
            SELECT *
            FROM studies
            WHERE study_id=@study_id
        )
        BEGIN
            THROW 53000, N'There is no study with given ID', 1
        END

        IF NOT EXISTS
        (SELECT p.status FROM payments p WHERE p.payment_id=@payment_id AND p.[status]=0 AND p.postponed=0)
        BEGIN
            THROW 53000, N'Cannot add the item to this payment',  1
        END

        IF dbo.calc_study_vacancy_amount(@study_id) <= 0
        BEGIN
            THROW 53000, N'The are no vacancies for this study', 1
        END
        INSERT INTO study_payments(study_id, payment_id)
        VALUES (@study_id, @payment_id)
    END TRY
    BEGIN CATCH
        DECLARE @msg NVARCHAR(2048) = N'ERROR: ' + ERROR_MESSAGE();
        THROW 53000, @msg, 1
    END CATCH
END
```

12. Wyświetl liczbę studentów zapisanych na wydarzenia odbywające się w danym przedziale czasu

Procedura wyświetlająca liczbę studentów, którzy są zapisani na wydarzenia w danym przedziale czasu.

```
CREATE PROCEDURE get_students_registered_count_in_period(@start_date DATE, @end_date DATE)
AS
BEGIN
    if(@start_date > @end_date)
    BEGIN
        THROW 53000, N'Start date must be earlier than end date', 1
    END
    SELECT *
    FROM students_registered_count src
    WHERE src.date > @start_date AND src.date < @end_date
END
```

Przykładowy rezultat procedury dla
@start_date = '2023-10-01',
@end_date = '2023-11-15'

lesson_id	name	date	count	lesson form
73	shorting lesson	2023-10-31	7	stationary
95	hundreding lesson	2023-10-15	21	stationary
96	ableing lesson	2023-10-30	21	stationary
97	thusing lesson	2023-11-14	21	stationary

13. Wyświetl raport dotyczący frekwencji na zakończonych już wydarzeniach w danym przedziale czasu

Procedura wyświetlająca liczbę studentów, którzy są zapisani na zakończone już wydarzenia w danym przedziale czasu.

```
CREATE PROCEDURE get_attendance_percentage_report_in_period(@start_date DATE, @end_date DATE)
AS
BEGIN
    if(@start_date > @end_date)
    BEGIN
        THROW 53000, N'Start date must be earlier than end date', 1
    END
    SELECT *
    FROM attendance_percentage_report apr
    WHERE apr.date > @start_date AND apr.date < @end_date
END
```

Przykładowy rezultat procedury dla
@start_date = '2023-10-01',
@end_date = '2023-11-15'

lesson_id	name	date	Attendance Percentage
73	shorting lesson	2023-10-31	1.00
95	hundreding lesson	2023-10-15	0.86
96	ableing lesson	2023-10-30	0.86
97	thusing lesson	2023-11-14	0.95

Raporty finansowe

14. Przychody dla studiów w zadanym okresie

Procedura wyświetlająca przychody dla wszystkich studiów w danym przedziale czasu.

```
CREATE PROCEDURE get_studies_income(
    @start_date date = '0001-01-01',
    @end_date date = '9999-12-31'
)
AS
BEGIN
    IF @start_date > @end_date
        BEGIN
            THROW 53000, N'start_date should be before end_date', 1
        END;

    ;WITH t AS (select lp.lesson_id, l.price, l.extended_price, s.study_id, lp.is_extended
                FROM lesson_payments lp
                         JOIN lessons l
                              ON lp.lesson_id = l.lesson_id
                         JOIN courses c
                              ON l.course_id = c.course_id
                         JOIN studies s
                              ON c.study_id = s.study_id
                         JOIN payments p
                              ON lp.payment_id = p.payment_id AND p.status = 1
                WHERE p.date >= @start_date
                  AND p.date <= @end_date)
    SELECT s.study_id,
           s.name,
           s.description,
           s.employee_id,
           ROUND((SELECT COUNT(*)
                  FROM study_payments sp
                           JOIN payments p
                                ON sp.payment_id = p.payment_id AND p.status = 1
                  WHERE sp.study_id = s.study_id
                    AND p.date >= @start_date
                    AND p.date <= @end_date) * s.entry_fee
                    +
                 (SELECT ISNULL(SUM(t.price * t.is_extended + t.extended_price * ABS(t.is_extended - 1)), 0)
                  FROM t
                  WHERE t.study_id = s.study_id)
                 , 2) AS income
    FROM studies s
END;
```

Przykładowy rezultat procedury

study_id	name	description	employee_id	income
1	amonging study	amonging study description	5	156170.0000
2	responsibilitying study	responsibilitying study description	5	63096.0000

15. Przychody za kursy w zadanym okresie

Procedura wyświetlająca przychody dla wszystkich kursów w danym przedziale czasu.

```
CREATE PROCEDURE get_courses_income(
    @start_date date = '0001-01-01',
    @end_date date = '9999-12-31'
)
AS
BEGIN
    IF @start_date > @end_date
        BEGIN
            THROW 53000, N'start_date should be before end_date', 1
        END

    ;WITH t AS (SELECT cp.course_id, cp.is_full_price
                FROM course_payments cp
                         JOIN payments p
                              ON cp.payment_id = p.payment_id AND p.status = 1
                WHERE p.date >= @start_date
                  AND p.date <= @end_date)
    SELECT c.course_id,
           c.name,
           c.description,
           ROUND((SELECT COUNT(*)
                  FROM t
                  WHERE t.course_id = c.course_id
                    AND t.is_full_price = 1) * c.full_price
                   +
                  (SELECT COUNT(*)
                   FROM t
                   WHERE t.course_id = c.course_id
                     AND t.is_full_price = 0) * c.entry_price, 2) AS income
    FROM courses c
    WHERE c.study_id IS NULL
END;
```

Przykładowy rezultat procedury

course_id	name	description	income
9	girling course	girling course description	48300.0000
15	pering course	pering course description	18160.0000
16	ining course	ining course description	7035.0000

16. Przychody za webinary w zadanym okresie

Procedura wyświetlająca przychody dla wszystkich webinarów w danym przedziale czasu.

```
CREATE PROCEDURE get_webinars_income(
    @start_date date = '0001-01-01',
    @end_date date = '9999-12-31'
)
AS
BEGIN
    IF @start_date > @end_date
        BEGIN
            THROW 53000, N'start_date should be before end_date', 1
        END;

    WITH t AS (SELECT l.lesson_id, COUNT(lp.lesson_id) as counter
               FROM lessons l
                        JOIN lesson_payments lp
                             ON l.lesson_id = lp.lesson_id
                        JOIN payments p
                             ON lp.payment_id = p.payment_id AND p.status = 1
               WHERE l.course_id IS NULL
                 AND p.date >= @start_date
                 AND p.date <= @end_date
               GROUP BY l.lesson_id)
    SELECT l.lesson_id,
           l.name,
           l.description,
           l.date,
           ROUND(t.counter * l.price, 2) AS income
    FROM t
             JOIN lessons l
                  ON l.lesson_id = t.lesson_id
END;
```

Przykładowy rezultat procedury

lesson_id	name	description	date	income
69	theorying lesson	theorying lesson description	2023-07-01	1199.8800
70	especiallying lesson	especiallying lesson description	2023-07-31	1349.8500
71	likelying lesson	likelying lesson description	2023-08-31	799.9000
72	pressureing lesson	pressureing lesson description	2023-09-30	599.9000
73	shorting lesson	shorting lesson description	2023-10-31	769.9300

17. Lista obecności w danym przedziale czasu

Procedura wyświetlająca listę obecności na wydarzeniach odbywających się w danym przedziale czasu.

```
CREATE PROCEDURE get_attendance_list_in_period(@start_date DATE, @end_date DATE)
AS
BEGIN
    if(@start_date > @end_date)
    BEGIN
        THROW 53000, N'Start date must be earlier than end date', 1
    END
    SELECT *
    FROM attendance_list al
    WHERE al.date > @start_date AND al.date < @end_date
END
```

Przykładowy rezultat procedury dla
@start_date = '2023-10-01',
@end_date = '2023-11-15'

lesson_id	date	first_name	last_name	status
73	2023-10-31	Alan	Lawson	PRESENT
73	2023-10-31	Justin	Vance	PRESENT
73	2023-10-31	Barbara	Briggs	PRESENT
73	2023-10-31	Thomas	Mullins	PRESENT
73	2023-10-31	Joshua	Moore	PRESENT
73	2023-10-31	Charles	Thomas	PRESENT

18. Raport bilokacji w danym przedziale czasu

Procedura wyświetlająca listę studentów, którzy mają kolidujące ze sobą lekcje w danym przedziale czasu.

```
CREATE PROCEDURE get_bilocation_report_in_period(@start_date DATE, @end_date DATE)
AS
BEGIN
    if(@start_date > @end_date)
    BEGIN
        THROW 53000, N'Start date must be earlier than end date', 1
    END
    SELECT *
    FROM bilocation_report br
    WHERE br.date >= @start_date AND br.date <= @end_date
END
```

Przykładowy rezultat procedury dla
@start_date = '2023-10-01',
@end_date = '2023-11-15'

student_id	date	lesson 1 id	lesson 1	lesson 2 id	lesson 2
65	2024-03-20	115	senseing lesson	105	clearlying lesson
66	2024-03-20	115	senseing lesson	105	clearlying lesson
67	2024-03-20	115	senseing lesson	105	clearlying lesson
68	2024-03-20	115	senseing lesson	105	clearlying lesson
69	2024-03-20	115	senseing lesson	105	clearlying lesson
70	2024-03-20	115	senseing lesson	105	clearlying lesson

19. Opłacanie wydarzeń

Procedura pozwalająca opłacić dane wydarzenie (jeśli są na nim jeszcze wolne miejsca)

```
CREATE PROCEDURE pay_for_event(@payment_id INT)
AS
BEGIN
    BEGIN TRY
        IF EXISTS (
            SELECT 1
            FROM payments p
            JOIN lesson_payments lp ON p.payment_id=lp.payment_id
            WHERE p.[payment_id]=@payment_id AND dbo.get_lesson_vacancy_count(lp.lesson_id) <= 0
            )
            THROW 53000, N'There are lessons with no vacancies in your order.', 1
        ELSE IF EXISTS (
            SELECT 1
            FROM payments p
            JOIN course_payments cp ON p.payment_id=cp.payment_id
            WHERE p.[payment_id]=@payment_id AND dbo.get_course_vacancy_count(cp.course_id) <= 0
            )
            THROW 53000, N'There are courses with no vacancies in your order.', 1
        ELSE IF EXISTS (
            SELECT 1
            FROM payments p
            JOIN study_payments sp ON p.payment_id=sp.payment_id
            WHERE p.[payment_id]=@payment_id AND dbo.get_study_vacancy_count(sp.study_id) <= 0
            )
            THROW 53000, N'There are studies with no vacancies in your order.', 1
        UPDATE payments
        SET status = 1
        WHERE payment_id = @payment_id;
    END TRY
    BEGIN CATCH
        DECLARE @msg NVARCHAR(2048) = N'ERROR: ' + ERROR_MESSAGE();
        THROW 53000, @msg, 1
    END CATCH
END
```

Funkcje

1. Obliczanie wolnych miejsc na danych studiach

Funkcja obliczająca ilość wolnych miejsc na danych studiach.

```
CREATE FUNCTION calc_study_vacancy_amount(@study_id INT)
    RETURNS INT
        AS
        BEGIN
            RETURN
            (
                SELECT s.students_limit - COUNT(*)
                FROM studies s
                JOIN study_payments sp ON s.study_id=sp.study_id
                JOIN payments p ON p.payment_id=sp.payment_id
                WHERE s.study_id=@study_id AND (p.[status]=1 OR p.postponed=1)
                GROUP BY s.students_limit
            )
        END
```

2. Obliczanie wolnych miejsc na danym kursie

Funkcja obliczająca ilość wolnych miejsc na danym kursie.

```
CREATE FUNCTION calc_course_vacancy_amount(@course_id INT)
    RETURNS INT
        AS
        BEGIN
            DECLARE @result INT
            DECLARE @course_study_id INT
            SET @course_study_id = (SELECT c.study_id FROM courses c WHERE c.course_id=@course_id)
            IF @course_study_id IS NOT NULL
                BEGIN
                    SET @result = dbo.calc_study_vacancy_amount(@course_study_id)
                END
            ELSE
                SET @result = (
                    SELECT
                        c.students_limit - COUNT(*)
                    FROM courses c
                    LEFT JOIN course_payments cp ON c.course_id=cp.course_id
                    LEFT JOIN payments p ON p.payment_id=cp.payment_id
                    WHERE c.course_id=@course_id AND (p.[status]=1 OR p.postponed=1)
                    GROUP BY c.students_limit
                )
            RETURN @result
        END
```

3. Obliczanie wolnych miejsc na danej lekcji

Funkcja obliczająca ilość wolnych miejsc na danej lekcji.

```
CREATE FUNCTION calc_lesson_vacancy_amount(@lesson_id INT)
    RETURNS INT
        AS
        BEGIN
            DECLARE @result INT
            DECLARE @MAXINT INT
            SET @MAXINT = 2147483647
            SET @result = (
                    SELECT l.students_limit - COUNT(*)
                    FROM lessons l
                    LEFT JOIN lesson_payments lp ON l.lesson_id=lp.lesson_id
                    LEFT JOIN payments p ON p.payment_id=lp.payment_id
                    WHERE l.lesson_id=@lesson_id AND (p.[status]=1 OR p.postponed=1)
                    GROUP BY l.students_limit
                )
            IF @result IS NULL
                BEGIN
                    SET @result = (SELECT students_limit FROM lessons WHERE lesson_id=@lesson_id)
                END
            IF @result IS NULL
                SET @result=@MAXINT
            RETURN @result
        END
```

4. Szukanie id studiów do których należy lekcja

Funkcja znajdująca id studiów do których należy dana lekcja, jeśli nie jest ona częścią studiów zwraca NULL.

```
CREATE FUNCTION get_lesson_studies(@lesson_id INT)
    RETURNS INT
        AS
        BEGIN
            DECLARE @lesson_course_id INT
            SET @lesson_course_id = (SELECT l.course_id FROM lessons l WHERE l.lesson_id=@lesson_id)
            IF @lesson_course_id IS NULL
                BEGIN
                    RETURN NULL
                END
            RETURN ( SELECT c.study_id FROM courses c WHERE c.course_id=@lesson_course_id)
        END
```

Triggery

1. Usuwanie zaliczek

Trigger usuwający zaliczkę po wpłynięciu płatności za całość kursu.

```
CREATE TRIGGER tr_remove_prev_partial_payments
ON course_payments
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    DELETE FROM course_payments
    WHERE EXISTS (
        SELECT 1
        FROM inserted i
        WHERE course_payments.course_id = i.course_id
          AND course_payments.payment_id = i.payment_id
          AND i.is_full_price = 1
    )
    AND is_full_price = 0;
END;
```

2. Trigger dodający wpis o obecności na zajęciach po zapłaceniu za te zajęcia

Trigger aktywuje się po zdarzeniu dodania recordu do tabeli payments ze statusem opłacone oraz po zaaktualizowaniu recordu w tabeli payments tylko przy zmianie statusu z nieopłacony na opłacony

```
CREATE TRIGGER tr_generate_attendance
    ON payments
    AFTER INSERT, UPDATE
    AS
BEGIN
    SET NOCOUNT ON
    DECLARE @paid_inserted TABLE (student_id INT, payment_id INT)

    IF NOT EXISTS(select * from deleted)
        BEGIN
            INSERT INTO @paid_inserted
            SELECT i.student_id, i.payment_id
            FROM inserted i
            WHERE i.status = 1
        END
    ELSE
        BEGIN
            INSERT INTO @paid_inserted
            SELECT i.student_id, i.payment_id
            FROM inserted i
                JOIN deleted d
                    ON d.payment_id = i.payment_id
            WHERE i.status = 1 and d.status = 0
        END

    INSERT INTO attendance (lesson_id, student_id)
    SELECT lp.lesson_id, p_inserted.student_id
    FROM @paid_inserted p_inserted
        JOIN lesson_payments lp
            ON lp.payment_id = p_inserted.payment_id

    INSERT INTO attendance (lesson_id, student_id)
    SELECT l.lesson_id, p_inserted.student_id
    from @paid_inserted p_inserted
        JOIN course_payments cp
            ON cp.payment_id = p_inserted.payment_id
        JOIN lessons l
            ON l.course_id = cp.course_id
END;
```

Role

1. Admin

```
CREATE ROLE admin
GRANT ALL PRIVILEGES ON u_ksliwins.dbo TO admin
```

2. Pracownik zarządzający wydarzeniami

```
CREATE ROLE event_manager

GRANT EXECUTE ON add_study_lesson TO event_manager
GRANT EXECUTE ON add_course_lesson TO event_manager
GRANT EXECUTE ON add_webinar TO event_manager
GRANT UPDATE ON lessons TO event_manager
GRANT DELETE ON lessons TO event_manager
GRANT INSERT ON courses TO event_manager
GRANT UPDATE ON courses TO event_manager
GRANT DELETE ON courses TO event_manager
GRANT INSERT ON studies TO event_manager
GRANT UPDATE ON studies TO event_manager
GRANT DELETE ON studies TO event_manager
GRANT INSERT ON students TO event_manager
GRANT UPDATE ON students TO event_manager
GRANT SELECT ON students_registered_count TO event_manager
GRANT SELECT ON students_registered_future_count TO event_manager
GRANT SELECT ON attendance_percentage_report TO event_manager
GRANT SELECT ON attendance_list TO event_manager
GRANT SELECT ON bilocation_report TO event_manager
GRANT EXEC ON get_attendance_list_in_period TO event_manager
GRANT EXEC ON get_attendance_percentage_report_in_period TO event_manager
GRANT EXEC ON get_students_registered_count_in_period TO event_manager
GRANT EXEC ON get_bilocation_report_in_period TO event_manager
```

3. Pracownik biurowy

```
CREATE ROLE office_worker

GRANT SELECT ON webinars_income TO office_worker
GRANT SELECT ON courses_income TO office_worker
GRANT SELECT ON studies_income TO office_worker
GRANT SELECT ON students_registered_count TO office_worker
GRANT SELECT ON debtors_list TO office_worker
GRANT SELECT ON webinars_debtors_list TO office_worker
GRANT SELECT ON studies_debtors_list TO office_worker
GRANT SELECT ON students_registered_count TO office_worker
GRANT SELECT ON students_registered_future_count TO office_worker
GRANT SELECT ON attendance_percentage_report TO office_worker
GRANT SELECT ON attendance_list TO office_worker
GRANT SELECT ON bilocation_report TO office_worker
GRANT EXEC ON get_attendance_list_in_period TO office_worker
GRANT EXEC ON get_attendance_percentage_report_in_period TO office_worker
GRANT EXEC ON get_students_registered_count_in_period TO office_worker
GRANT EXEC ON get_bilocation_report_in_period TO office_worker
```

4. Wykładowca

```
CREATE ROLE lecturer

GRANT EXECUTE update_attendance TO lecturer
GRANT INSERT ON attendance TO lecturer
GRANT UPDATE ON attendance TO lecturer
GRANT INSERT ON materials TO lecturer
GRANT UPDATE ON materials TO lecturer
GRANT DELETE ON materials TO lecturer
GRANT SELECT ON attendance_list TO lecturer
GRANT EXEC ON get_attendance_list_in_period TO office_worker
```

5. Student

```
CREATE ROLE student

GRANT EXECUTE ON add_lesson_to_cart TO student
GRANT EXECUTE ON add_course_to_cart TO student
GRANT EXECUTE ON add_study_to_cart TO student
GRANT EXECUTE ON student_attendance_history TO student
GRANT EXECUTE ON student_cart_lessons_info TO student
GRANT EXECUTE ON student_cart_courses_info TO student
GRANT EXECUTE ON student_cart_studies_info TO student
GRANT SELECT ON materials TO student
GRANT EXEC ON pay_for_event TO student
```

Indeksy

Klucze główne oraz wartości unikalne email jest wartością unikalną, ponieważ wymuszamy na studentach, aby każdy z nich miał indywidualny email. file_url jest wartością unikalną, ponieważ musimy mieć możliwość otrzymania dostępu do każdego materiału osobno. payment_url jest wartością unikalną, ponieważ każda płatność musi mieć swój osobny url, który ją identyfikuje.

```
-- Klucze głowne
CREATE UNIQUE INDEX students_pk
ON students (student_id)

CREATE UNIQUE INDEX employees_pk
ON employees (employee_id)

CREATE UNIQUE INDEX lecturers_pk
ON lecturers (lecturer_id)

CREATE UNIQUE INDEX studies_pk
ON studies (study_id)

CREATE UNIQUE INDEX courses_pk
ON courses (course_id)

CREATE UNIQUE INDEX lessons_pk
ON lessons (lesson_id)

CREATE UNIQUE INDEX payments_pk
ON payments (payment_id)

CREATE UNIQUE INDEX materials_pk
ON materials (material_id)

CREATE UNIQUE INDEX internships_pk
ON internships (internship_id)

CREATE UNIQUE INDEX roles_pk
ON roles (role_id);

CREATE UNIQUE INDEX attendance_pk
ON attendance (student_id, lesson_id)

CREATE UNIQUE INDEX study_payments_pk
ON study_payments (study_id, payment_id)

CREATE UNIQUE INDEX course_payments_pk
ON course_payments (course_id, payment_id)

CREATE UNIQUE INDEX lesson_payments_pk
ON lesson_payments (lesson_id, payment_id)

-- Emaile
CREATE UNIQUE INDEX students_email
ON students (email)

CREATE UNIQUE INDEX lecturers_email
ON lecturers (email)

CREATE UNIQUE INDEX employees_email
ON employees (email)

-- File Url
CREATE UNIQUE INDEX file_url
ON materials (file_url)

-- Payment Url
CREATE UNIQUE INDEX payment_url
ON payments (payment_url)
```