Wojskowa Akademia Techniczna
im. Jarosława Dąbrowskiego

Wydział Cybernetyki, kierunek informatyka - inżynieria systemów

Realizacja zadania laboratoryjnego w ramach przedmiotu:

Systemy Baz Danych

Temat laboratorium:

Temporalne Bazy Danych

Opracował: Radosław Relidzyński, **Grupa:** WCY23IX3S4

Spis treści

Wstęp teoretyczny

Baza danych – „uporządkowany zbiór danych określających wybrany fragment rzeczywistości lub problemu, które są przechowywane trwale w pamięci komputerowej do której może mieć dostęp wielu użytkowników w dowolnej chwili czasu.”

System zarządzania bazami danych – „zorganizowany zbiór narzędzi (programów komputerowych i bibliotek), które umożliwiają wykonanie podstawowych operacji na danych (CRUD) zawartych w jednej lub więcej bazach danych.”

System baz danych – jego definicja wyraża się wzorem:

$$SBD =< \{U, SO, DB, SZBD, P\}, R >$$

Gdzie:

$$U - zbiór\ urządzeń$$

$$SO - system\ operacyjny$$

$$BD - baza\ danych\ (schemat, stan, ścieżki\ dostępu)$$

$$SZBD - system\ zarządzania\ bazą\ danych$$

$$P - polecenia\ użytkownika$$

$$R - relacje\ między\ obiektami\ SBD\ a\ otoczeniem$$

[źródło: materiały z wykładu „Temporalne bazy danych” dr inż. Jarosława Koszeli]

Środowisko

Opis środowiska

W ramach projektu rolę systemu zarządzania bazą danych będzie pełnić narzędzie Microsoft SQL Server, a baza danych zostanie utworzona pod nazwą „lotniska europy.” Dodatkowo wykorzystane zostanie środowisko SQL Server Management Studio.

SQL Server Management Studio – „zintegrowane środowisko do zarządzania wszystkimi komponentami (baza danych, usługi analityczne, usługi raportowe itd.), wchodzącymi w skład Microsoft SQL Server. Zawiera narzędzia do konfiguracji, monitorowania i administrowania instancjami SQL Server. Umożliwia budowę zapytań i skryptów, zawiera zarówno edytor skryptów jak i narzędzia graficzne.”
[źródło: https://pl.wikipedia.org/wiki/SQL_Server_Management_Studio]

Microsoft SQL Server – „system zarządzania bazą danych, wspierany i rozpowszechniany przez korporację Microsoft. Jest to główny produkt bazodanowy tej firmy, który charakteryzuje się tym, iż jako język zapytań używany jest przede wszystkim Transact-

SQL, który stanowi rozwinięcie standardu ANSI/ISO.”
[źródło: https://pl.wikipedia.org/wiki/Microsoft_SQL_Server]

Wykaz informacji o wersjach środowiska:

System operacyjny: Windows 11 Pro 23H2

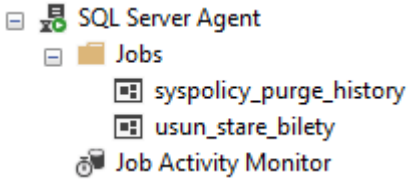
Informacja o wersjach komponentów SQL Server Management Studio:

SQL Server Management Studio	20.1.10.0
SQL Server Management Objects (SMO)	17.100.31.0+ccc8de5aee492d8b603963ce96e9f9877a21afa4
Microsoft T-SQL Parser	17.2.1.1+f4dfdb798c9aaa90bbe9a0cbfd92e100200ad26c.f4dfdb798c9aaa90bbe9a0cbfd92e100200ad26c
Microsoft Analysis Services Client Tools	20.0.3.0
Microsoft Data SqlClient (MDS)	5.1.5
Microsoft SQL Server Data-Tier Application Framework (DacFX)	162.2.111.2+1a4d708fee9d82fe4e01e02f3962d1e83d374807.1a4d708fee9d82fe4e01e02f3962d1e83d374807
Microsoft .NET Framework	4.0.30319.42000
Operating System	10.0.22631

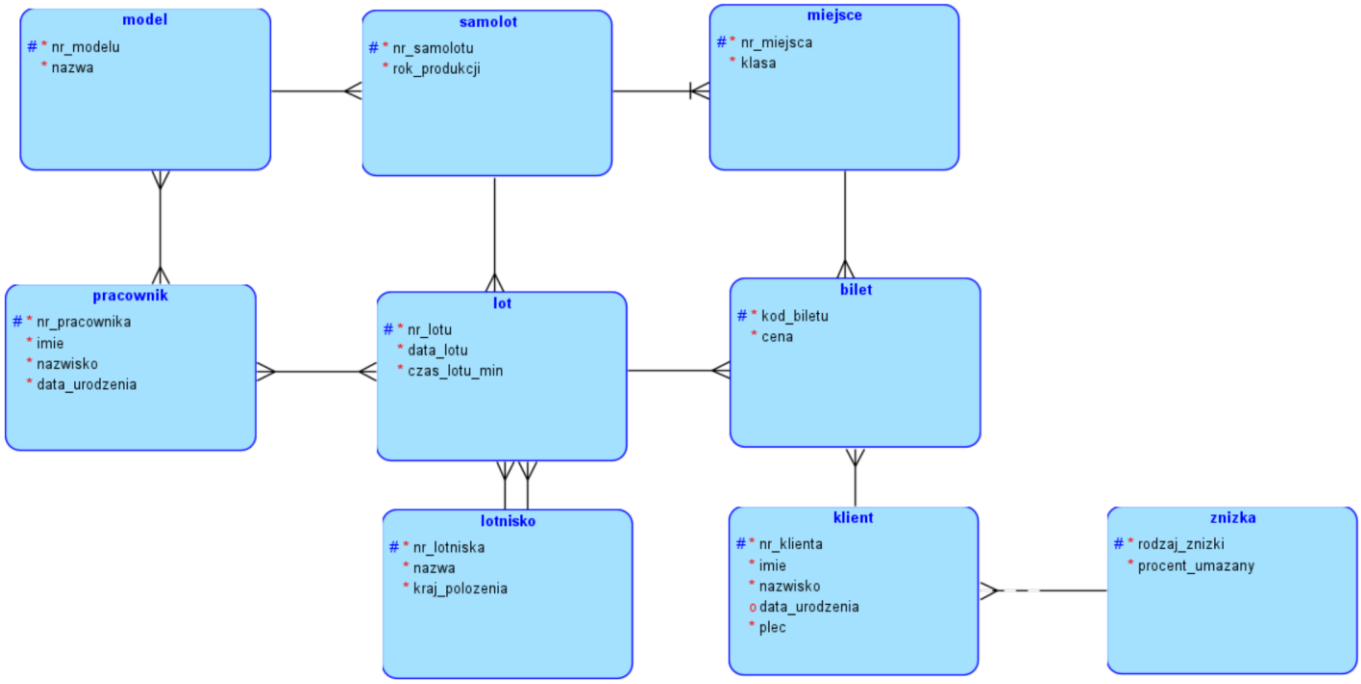
Instrukcja do przygotowania bazy danych

1. **Inicjalizacja bazy – uruchomienie skryptu „create.sql”.**
2. **Wypełnienie bazy danymi – uruchomienie skryptu „fill_data.sql”.**
3. **Sprawdzenie czy baza jest wypełniona – uruchomienie skryptu „check.sql”.**
4. **Wywołanie przykładowych zapytań – uruchomienie skryptu „query.sql”.**
5. **Dodanie i sprawdzenie wyzwalaczy:**
    a. Dodanie pierwszego – uruchomienie skryptu „trg_urodziny.sql”.
    b. Sprawdzenie pierwszego – uruchomienie skryptu „test_trg_ urodziny.sql”.
    c. Dodanie drugiego – uruchomienie skryptu „trg_powrot_klienta.sql”.
    d. Sprawdzenie drugiego – uruchomienie skryptu „test_trg_powrot_klienta.sql”.
    e. Dodanie trzeciego – uruchomienie skryptu „trg_licznik_lotow.sql”.
    f. Sprawdzenie trzeciego – uruchomienie skryptu „test_trg_licznik_lotow.sql”.
    g. Ustawienie priorytetu wyzwalaczy – uruchomienie skryptu „set_trg_priority.sql”.
6. **Dodanie procedury:**
    a. Stworzenie procedury – uruchomienie skryptu „prd_usun_stare_bilety.sql”.

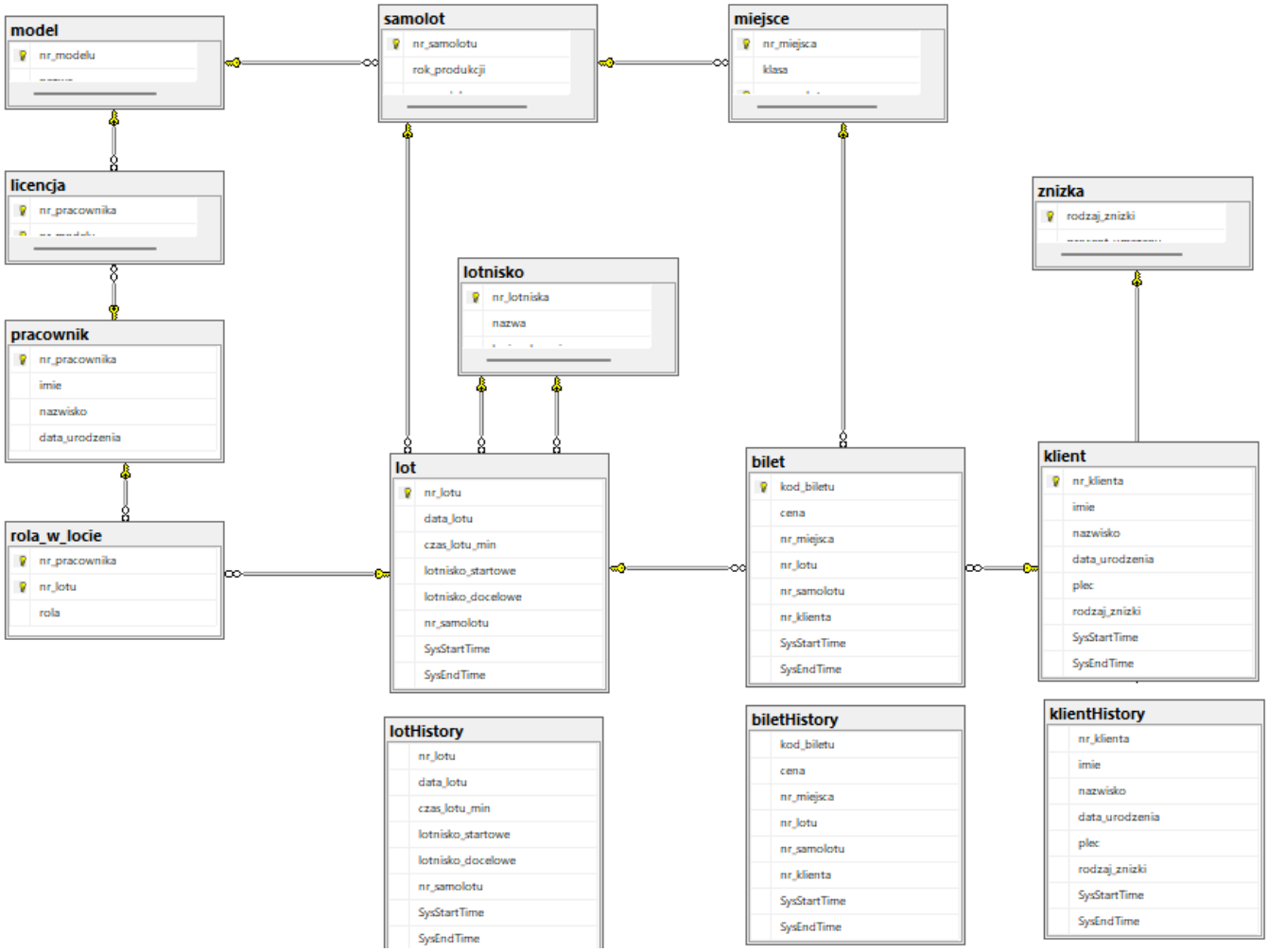
b. Uruchomienie agenta
    i. W eksploratorze rozwinąć opcje dla serwera.
    ii. Kliknąć prawym przyciskiem myszy na „SQL Server Agent” i wybrać opcję „start” (może być wymagane potwierdzenie jako administratora).
c. Dodanie zadania wykonującego prodedurę:
    i. Kliknąć ponownie prawym przyciskiem myszy na „SQL Server Agent”, wybrać opcję „New”, następnie „Job…”.
    ii. Podać nazwę, na przykład „usun_stare_bilety”.
    iii. Przejść do zakładki „Steps” i wykonać następujące działania:
        1. Podać nazwę, na przykład „usun_stare_bilety”.
        2. W polu „Database” wybrać z listy rozwijanej „lotniska_europy”.
        3. W polu „Command” wpisać instrukcję wywołania procedury, czyli „EXEC prd_usun_stare_bilety”.
        4. Kliknąć „OK”.
    iv. Przejść do zakładki „Schedules” i wykonać następujące działania:
        1. Wybrać opcję „New…”.
        2. Podać nazwę, na przykład „usun_stare_bilety”.
        3. W ramach segmentu „Frequency” dla opcji „Occurs” wybrać z lisy rozwijanej opcję „Daily”.
        4. W ramach segmentu „Daily frequency” dla opcji „Occurs once at” zmienić, żeby uruchamiał się o godzinie „12:00:00 PM”.
        5. Kliknąć „OK”.
    v. Zakończyć dodawanie procedury klikając „OK”.
d. Sprawdzić, czy procedura jest dodana
    i. Rozwinąć opcje dla „SQL Server Agent”, a następnie „Jobs”,
    ii. Sprawdzić, czy lista zawiera „usun_stare_bilety”, poniżej przykład:



Model logiczny



Model relacyjny



Wykaz tabel

Baza danych stworzona w ramach projektu pełni zadanie zbierania i zarządzania informacjami dotyczącymi wybranych lotnisk Europy oraz lotów między nimi. W ramach tego powołane zostają poniższe tabele:

- Samolot – informacje o samolocie
- Miejsce – informacje o miejscu w samolocie
- Model – informacje o modelach samolotów
- Pracownik – Informacje o pracownikach
- Licencja – intersekcja między pracownikiem i modelem, mówi, jakim
- samolotem może lecieć dany pracownik
- Lotnisko – informacje o lotniskach
- Lot (*temporalna*) – informacje o locie na podstawie samolotu i lotnisk
- Rola w locie – intersekcja między pracownikiem i lotem, mówi, jaką
- rolę pełni dany pracownik w danym locie
- Klient (*temporalna*) – informacje o kliencie
- Zniżki – informacje o zniżkach klientów
- Bilet (*temporalna*) – informacje o bilecie na podstawie lotu, miejsca w samolocie oraz klienta

Wykaz zapytań

Zapytanie 1: suma cen biletów dla zniżek

Kod zapytania

```
-- suma cen biletów zakupionych w ramach różnych rodzajów zniżek przez klientów w 2023 roku
licząc tylko te o sumie większej niż 500
SELECT k.rodzaj_znizki, SUM(b.cena) AS suma_cen_biletow
FROM bilet FOR SYSTEM_TIME BETWEEN '2023-01-01' AND '2024-01-01' b
JOIN klient k ON b.nr_klienta = k.nr_klienta
GROUP BY k.rodzaj_znizki
HAVING SUM(b.cena) > 500
ORDER BY suma_cen_biletow DESC;
```

Rezultat zapytania

Results | Messages

	rodzaj_znizki	suma_cen_biletow
1	rodzinna	1453
2	NULL	944

Zapytanie 2: wiek lotu

Kod zapytania

```
-- wiek informacji o locie
SELECT l.nr_lotu, MIN(l.SysStartTime) AS data_dodania, DATEDIFF(DAY, MIN(l.SysStartTime),
GETDATE()) AS wiek_w_dniach
```

```
FROM lot FOR SYSTEM_TIME ALL l
GROUP BY l.nr_lotu;
```

Rezultat zapytania

Results | Messages

	nr_lotu	data_dodania	wiek_w_dniach
1	1	2020-10-11 12:00:00.0000000	1340
2	2	2020-10-11 12:00:00.0000000	1340
3	3	2020-10-11 12:00:00.0000000	1340
4	4	2020-10-11 12:00:00.0000000	1340
5	5	2020-10-11 12:00:00.0000000	1340
6	6	2020-10-11 12:00:00.0000000	1340
7	7	2020-10-11 12:00:00.0000000	1340
8	8	2020-10-11 12:00:00.0000000	1340
9	9	2020-10-11 12:00:00.0000000	1340
10	10	2020-10-11 12:00:00.0000000	1340

Zapytanie 3: średni czas lotu

Kod zapytania

```
-- średni czas dla lotu na podstawie wszystkich wartości historycznych
SELECT l.nr_lotu, AVG(l.czas_lotu_min) AS sredni_czas_lotu
FROM lot FOR SYSTEM_TIME ALL l
GROUP BY l.nr_lotu
ORDER BY sredni_czas_lotu DESC;
```

Rezultat zapytania

Results | Messages

	nr_lotu	sredni_czas_lotu
1	2	165
2	3	150
3	1	150
4	6	150
5	5	105
6	4	90
7	8	75
8	10	75
9	9	60
10	7	60

Zapytanie 4: liczba zmian w biletach lotu

Kod zapytania

```
-- zliczanie zmian w biletach w 2020 roku w zależności od lotu
SELECT b.nr_lotu, COUNT(*) AS liczba_zmian
FROM bilet FOR SYSTEM_TIME BETWEEN '2020-01-01' AND '2020-12-31' b
WHERE b.SysEndTime != '9999-12-31 23:59:59.9999999'
GROUP BY b.nr_lotu
ORDER BY liczba_zmian DESC;
```

Rezultat zapytania

Results Messages

	nr_lotu	liczba_zmian
1	1	4
2	2	3
3	3	2

Zapytanie 5: bilety kupione przez klienta w danym roku

Kod zapytania

```
-- zliczanie biletów kupionych przez klienta w zależności od roku
SELECT k.nr_klienta, CONCAT(k.imie, ' ', k.nazwisko) AS klient, YEAR(b.SysStartTime) AS Rok,
COUNT(DISTINCT b.kod_biletu) AS liczba_biletow
FROM bilet FOR SYSTEM_TIME ALL b
JOIN klient k ON b.nr_klienta = k.nr_klienta
GROUP BY k.nr_klienta, k.imie, k.nazwisko, YEAR(b.SysStartTime)
ORDER BY k.nr_klienta, Rok;
```

Rezultat zapytania

Results Messages

	nr_klienta	klient	Rok	liczba_biletow
1	1	Kamil Madajski	2020	3
2	2	Emilia Ksiazkiewicz	2019	3
3	2	Emilia Ksiazkiewicz	2020	4
4	3	Mateusz Dybek	2020	3
5	4	Zofia Komosa	2019	1
6	4	Zofia Komosa	2020	1
7	5	Krystyna Czarniak	2019	2
8	5	Krystyna Czarniak	2020	5
9	6	Janusz Grzelczak	2020	1
10	7	Albert Swiatkowski	2019	2
11	7	Albert Swiatkowski	2020	3
12	8	Krzysztof Cieluch	2020	4
13	9	Malgorzata Handzlik	2019	1
14	9	Malgorzata Handzlik	2020	2
15	10	Tobiasz Ciszek	2018	1
16	10	Tobiasz Ciszek	2020	4

Zapytanie 6: średnia cen biletów dla modelu

Kod zapytania

```
-- średnia cen biletów uwzględniając wszystkie aktualizacje w zależności od modelu samolotu
realizującego lot
SELECT m.nazwa AS ModelSamolotu, AVG(b.cena) AS srednia_cen_biletow
FROM bilet FOR SYSTEM_TIME ALL b
JOIN lot l ON b.nr_lotu = l.nr_lotu
JOIN samolot s ON l.nr_samolotu = s.nr_samolotu
JOIN model m ON s.nr_modelu = m.nr_modelu
GROUP BY m.nazwa
ORDER BY srednia_cen_biletow DESC;
```

Rezultat zapytania

Results | Messages

	ModelSamolotu	srednia_cen_biletow
1	Antonov AN-225 Mrija	116
2	C-54D Skymaster	103
3	Boeing 787-8 Dreamliner	99
4	A380-800 British Airways	96
5	Boeing 747-200	94
6	Concorde	77

Zapytanie 7: średnia cen biletów w zależności od zniżki historycznie

Kod zapytania

```
-- średnia cena biletów dla klienta w zależności od zniżki jaką ma lub miał
WITH ZmianyZnizek AS (
    SELECT k.nr_klienta, CONCAT(k.imie, ' ', k.nazwisko) AS klient, k.rodzaj_znizki,
k.SysStartTime, k.SysEndTime
    FROM klient FOR SYSTEM_TIME BETWEEN '2021-01-01' AND '2023-01-01' k
),
BiletyCeny AS (
    SELECT b.nr_klienta, b.cena, b.SysStartTime
    FROM bilet FOR SYSTEM_TIME BETWEEN '2021-01-01' AND '2023-01-01' b
)
SELECT zz.klient klient, zz.rodzaj_znizki, AVG(bc.cena) AS srednia_cen_biletow
FROM ZmianyZnizek zz
JOIN BiletyCeny bc ON zz.nr_klienta = bc.nr_klienta
GROUP BY zz.klient, zz.rodzaj_znizki
ORDER BY klient, rodzaj_znizki DESC;
```

Rezultat zapytania

Results | Messages

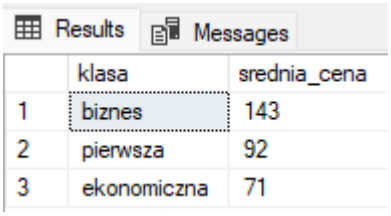
	klient	rodzaj_znizki	srednia_cen_biletow
1	Albert Swiatkowski	krwiodawca	65
2	Emilia Ksiazkiewicz	rodzinna	121
3	Emilia Ksiazkiewicz	NULL	121
4	Janusz Grzelczak	student	60
5	Kamil Madajski	student	91
6	Krystyna Czarniak	NULL	111
7	Krzysztof Cieluch	rodzinna	113
8	Malgorzata Handzlik	NULL	81
9	Mateusz Dybek	NULL	74
10	Tobiasz Ciszek	rodzinna	128

Zapytanie 8: średnia cen biletów na podstawie miejsc z danego momentu w czasie

Kod zapytania

```
-- średnia cena biletu na podstawie klas miejsc
SELECT m.klasa, AVG(b.cena) AS srednia_cena
FROM bilet FOR SYSTEM_TIME AS OF '2021-01-01 12:00:00.0000000' b
JOIN miejsce m ON b.nr_miejsca = m.nr_miejsca AND b.nr_samolotu = m.nr_samolotu
GROUP BY m.klasa
ORDER BY srednia_cena DESC;
```

Rezultat zapytania



Results | Messages

	klasa	srednia_cena
1	biznes	143
2	pierwsza	92
3	ekonomiczna	71

Wykaz wyzwalaczy

Wyzwalacz informacji, jeśli nowy klient założył konto w dniu urodzin

Uzasadnienie biznesowe: W ramach akcji promocyjnej klienci, którzy założą konto w dniu swoich urodzin mogą otrzymać dodatkową zniżkę.

```
-- jesli klient zalozy konto w dniu urodzin, to dostanie znizke
CREATE TRIGGER trg_urodziny
ON klient
AFTER INSERT
AS
BEGIN
    DECLARE @Message NVARCHAR(1000);

       DECLARE @nr_klienta INT;
       DECLARE @imie NVARCHAR(50);
    DECLARE @nazwisko NVARCHAR(50);
       -- DECLARE @data_dodania DATE;
       DECLARE @data_urodzenia DATE;
       DECLARE @plec NVARCHAR(1);
       DECLARE @rodzaj_znizki NVARCHAR(20);

    DECLARE @miesiac_dodania INT;
    DECLARE @dzien_dodania INT;
    DECLARE @miesiac_urodzenia INT;
    DECLARE @dzien_urodzenia INT;
       DECLARE @klient NVARCHAR(40);

       DECLARE client_birthday_cursor CURSOR FOR
    SELECT imie, nazwisko, data_urodzenia
    FROM inserted;

       OPEN client_birthday_cursor;
    FETCH NEXT FROM client_birthday_cursor INTO @imie, @nazwisko, @data_urodzenia;

       WHILE @@FETCH_STATUS = 0
    BEGIN
        SET @miesiac_dodania = MONTH(GETDATE());
        SET @dzien_dodania = DAY(GETDATE());
        SET @miesiac_urodzenia = MONTH(@data_urodzenia);
        SET @dzien_urodzenia = DAY(@data_urodzenia);
              SET @klient = CONCAT(@imie, ' ', @nazwisko);

        IF @miesiac_dodania = @miesiac_urodzenia AND @dzien_dodania = @dzien_urodzenia
        BEGIN
            SET @Message = FORMATMESSAGE('%s zalozyl wlasnie konto w dniu urodzin, przeslij mu
znizki', @klient);
            RAISERROR(@Message, 0, 1) WITH NOWAIT;
        END

        FETCH NEXT FROM client_birthday_cursor INTO @imie, @nazwisko, @data_urodzenia;
    END;
```

```
        CLOSE client_birthday_cursor;
    DEALLOCATE client_birthday_cursor;

END;
GO
```

Przykład zapytania aktywującego wyzwalacz:

```
-- TEST trg_urodziny
INSERT INTO KLIENT (IMIE, NAZWISKO, DATA_URODZENIA, RODZAJ_ZNIZKI, PLEC)
VALUES ('Dzisiaj', 'Urodzony', CAST(DATEADD(YEAR, -18, GETDATE()) AS DATETIME), NULL, 'M');

INSERT INTO KLIENT (IMIE, NAZWISKO, DATA_URODZENIA, RODZAJ_ZNIZKI, PLEC)
VALUES
       ('Anna', 'Kowalska', CAST(DATEADD(YEAR, -40, GETDATE()) AS DATETIME), NULL, 'M'),
       ('Jan', 'Nowak', CAST(DATEADD(YEAR, -30, GETDATE()) AS DATETIME), NULL, 'M'),
       ('Maria', 'Wi?niewska', CAST(DATEADD(YEAR, -20, GETDATE()) AS DATETIME), NULL, 'M'),
       ('Piotr', 'Zieli?ski', CAST(DATEADD(YEAR, -30, GETDATE()) AS DATETIME), NULL, 'M');
```

Messages

```
Dzisiaj Urodzony zalozyl wlasnie konto w dniu urodzin, przeslij mu znizki

(1 row affected)
Piotr Zieli?ski zalozyl wlasnie konto w dniu urodzin, przeslij mu znizki
Maria Wi?niewska zalozyl wlasnie konto w dniu urodzin, przeslij mu znizki
Jan Nowak zalozyl wlasnie konto w dniu urodzin, przeslij mu znizki
Anna Kowalska zalozyl wlasnie konto w dniu urodzin, przeslij mu znizki

(4 rows affected)

Completion time: 2024-06-01T02:33:17.8684189+02:00
```

Wyzwalacz sprawdzający czy założony użytkownik posiadał wcześniej już konto

Uzasadnienie biznesowe: Aby zoptymalizować nr_klienta tak, żeby mniej rosnął, po dodaniu ponownie tego samego klienta baza danych na podstawie tabeli historycznej przywraca jego stary numer.

```
-- jesli klient znajduje sie w tabeli historycznej, uzyj starego nr_klienta
CREATE TRIGGER trg_powrot_klienta
ON klient
AFTER INSERT
AS
BEGIN
    DECLARE @Message NVARCHAR(1000);

    DECLARE @nr_klienta INT;
    DECLARE @imie NVARCHAR(50);
    DECLARE @nazwisko NVARCHAR(50);
    DECLARE @data_urodzenia DATE;
    DECLARE @plec NVARCHAR(1);
       DECLARE @rodzaj_znizki NVARCHAR(20);

    DECLARE client_comeback_cursor CURSOR FOR
    SELECT imie, nazwisko, data_urodzenia, plec, rodzaj_znizki
    FROM inserted;
```

```
    OPEN client_comeback_cursor;
    FETCH NEXT FROM client_comeback_cursor INTO @imie, @nazwisko, @data_urodzenia, @plec,
@rodzaj_znizki;

    WHILE @@FETCH_STATUS = 0
    BEGIN
        SELECT @nr_klienta = nr_klienta
        FROM klientHistory
        WHERE imie = @imie AND nazwisko = @nazwisko AND data_urodzenia = @data_urodzenia;

        IF @nr_klienta IS NOT NULL
        BEGIN
            DELETE FROM klient
            WHERE imie = @imie AND nazwisko = @nazwisko AND data_urodzenia = @data_urodzenia;

                    SET IDENTITY_INSERT klient ON;

            INSERT INTO klient (nr_klienta, imie, nazwisko, data_urodzenia, plec, rodzaj_znizki)
            VALUES (@nr_klienta, @imie, @nazwisko, @data_urodzenia, @plec, @rodzaj_znizki);

            SET @Message = FORMATMESSAGE('Klient %s %s istnieje w historii, uzyto starego
nr_klienta %d.', @imie, @nazwisko, @nr_klienta);
            RAISERROR(@Message, 0, 1) WITH NOWAIT;
        END
        ELSE
        BEGIN
            SET @Message = FORMATMESSAGE('Klient %s %s zostal dodany jako nowy klient.', @imie,
@nazwisko);
            RAISERROR(@Message, 0, 1) WITH NOWAIT;
        END

        FETCH NEXT FROM client_comeback_cursor INTO @imie, @nazwisko, @data_urodzenia, @plec,
@rodzaj_znizki;
    END;

    CLOSE client_comeback_cursor;
    DEALLOCATE client_comeback_cursor;

END;
GO
```

Przykład sytuacji wywołania:

```
-- TEST trg_powrot_klienta
INSERT INTO KLIENT (IMIE, NAZWISKO, DATA_URODZENIA, RODZAJ_ZNIZKI, PLEC)
VALUES ('Karol', 'Karolewski', CAST('03-08-2002' AS DATETIME), NULL, 'M');

-- pobierz nr_klienta nowo dodanego klienta
DECLARE @nr_klienta INT;
SELECT @nr_klienta = nr_klienta
FROM KLIENT
WHERE IMIE = 'Karol' AND NAZWISKO = 'Karolewski' AND DATA_URODZENIA = CAST('03-08-2002' AS
DATETIME);

-- usun klienta
DELETE FROM KLIENT
WHERE nr_klienta = @nr_klienta;

-- dodaj ponownie klienta z nowymi danymi
INSERT INTO KLIENT (IMIE, NAZWISKO, DATA_URODZENIA, RODZAJ_ZNIZKI, PLEC)
VALUES ('Karol', 'Karolewski', CAST('03-08-2002' AS DATETIME), NULL, 'M');
```

Wyzwalacz zwracający licznik zmian w locie

Uzasadnienie biznesowe: W celu weryfikacji kolejnych zmian wyzwalacz rejestruje kolejne zmiany, może on potencjalnie również informować administratora o tym, że zaszły takie zmiany.

```
-- nie mozna edytowa? informacji o locie, który istnieje juz od miesiaca
CREATE TRIGGER trg_licznik_lotow
ON lot
AFTER UPDATE
AS
BEGIN
	DECLARE @Message NVARCHAR(1000);

    DECLARE @nr_lotu INT;
    DECLARE @licznik INT;

    SELECT @nr_lotu = INSERTED.nr_lotu
    FROM INSERTED;

    SELECT @licznik = COUNT(*)
    FROM dbo.lotHistory
    WHERE nr_lotu = @nr_lotu;

	SET @Message = FORMATMESSAGE('To jest %s zmiana lotu numer  %s ', CAST(@licznik AS
VARCHAR), CAST(@nr_lotu AS VARCHAR));
	RAISERROR(@Message, 0, 1) WITH NOWAIT;

END;
GO
```

Przykład sytuacji wywołania:

```
-- TEST trg_licznik_lotow
INSERT INTO lot (data_lotu, czas_lotu_min, lotnisko_startowe, lotnisko_docelowe, nr_samolotu)
VALUES (DATEADD(MONTH, 1, CONVERT(DATE, GETDATE())), 120, 1, 2, 1);

-- pobierz nr_lotu nowo dodanego lotu
DECLARE @nr_lotu INT;
SELECT @nr_lotu = nr_lotu
```

```
FROM lot
WHERE data_lotu = DATEADD(MONTH, 1, CONVERT(DATE, GETDATE())) AND nr_samolotu = 1;

UPDATE lot
SET czas_lotu_min = 130
WHERE nr_lotu = @nr_lotu;

UPDATE lot
SET czas_lotu_min = 140
WHERE nr_lotu = @nr_lotu;

UPDATE lot
SET czas_lotu_min = 150
WHERE nr_lotu = @nr_lotu;

DELETE FROM lot
WHERE data_lotu = DATEADD(MONTH, 1, CONVERT(DATE, GETDATE())) AND nr_samolotu = 1;
```

Messages

```
(1 row affected)
To jest 1 zmiana lotu numer  12

(1 row affected)
To jest 2 zmiana lotu numer  12

(1 row affected)
To jest 3 zmiana lotu numer  12

(1 row affected)

(1 row affected)

Completion time: 2024-06-01T02:35:54.5610151+02:00
```

Wykaz procedur

Procedura usuwająca bilety na loty, które już się odbyły

```
CREATE PROCEDURE prd_usun_stare_bilety
AS
BEGIN
    DELETE b
    FROM bilet b
    JOIN lot l ON b.nr_lotu = l.nr_lotu
    WHERE l.data_lotu < CAST(GETDATE() AS DATE);
END;
```

Podsumowanie

W ramach realizacji zadania laboratoryjnego udało się stworzyć system zarządzania informacjami w zakresie lotnisk Europy.

Udało się skutecznie wykorzystać narzędzia Microsoft SQL Server oraz SQL Server Management Studio. W projekcie wykorzystano zaawansowane funkcje takie jak wyzwalacze i procedury

Wprowadzenie temporalności niektórych tabel umożliwia przechowywanie i analizę danych w kontekście ich zmian w czasie, co pozwala na większą kontrolę w zakresie integralności i kontroli zmian w bazie.