Eksamensopgave
Databaser for udviklere

PBA Softwareudvikling

Lars Søndergaard Petersen

Indholdsfortegnelse

Introduktion

Opgaven går ud på at lav et bookingsystem til en hotelkæde som udlejer til virksomhedsarrangementer og kurser. Der er stillet nogle opgaver og krav som skal overholdes, de kan findes i opgaveformuleringen.

For at øge udfordringen lidt vil jeg forsøge at undgå cursors og andre loops i sql scriptet. Query optimizeren kan dog stadig vælge at løse en opgave med et loop.

Opgaven må max være 8 sider så de fleste kode eksempler skal findes i bilag.

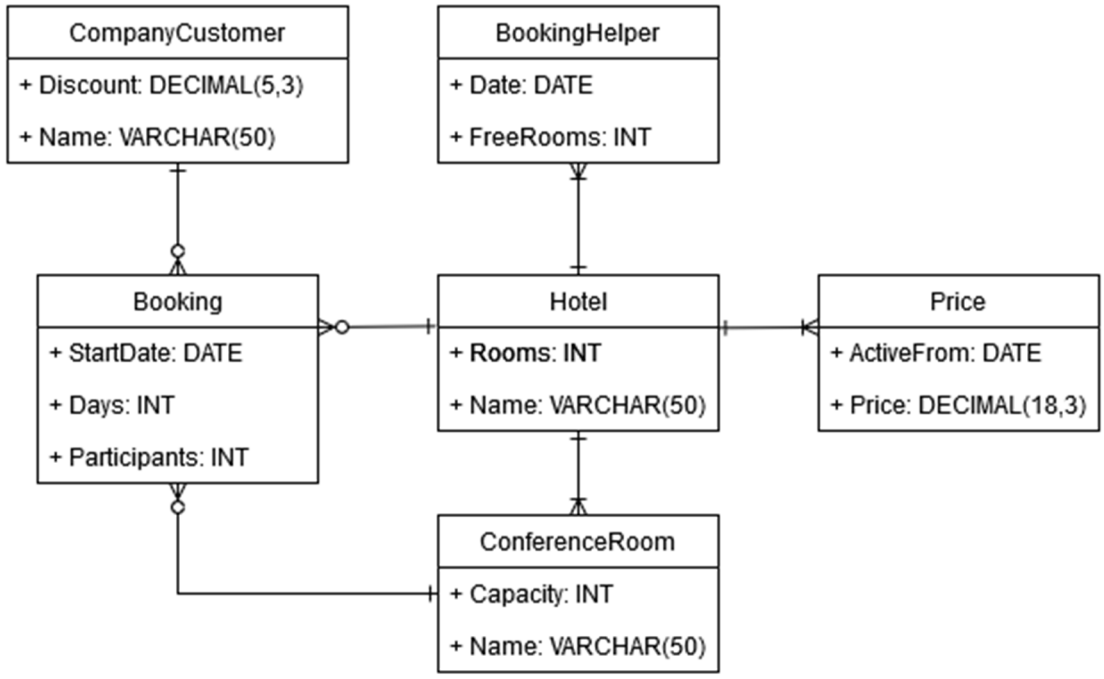
Alt kode til denne opgave ligger på Github, det vil dog være privat til afleveringsfristen er ovre:

https://github.com/trezum/database_exam

1 Klassediagram og tabeldesign

Klassediagram

Her ses det klasse diagram jeg er nået frem til på baggrund af de krav der stilles i opgaven:



Her er et par kommentarer til at uddybe de mindre tydelige dele af diagrammet.

Der kan argumenteres for redundans ved at have en fremmednøgle for hotel på en booking, men fordi en booking i særlig grad høre til et hotel syntes jeg stadig det giver mening.

Grunden til at til at en booking kan have et ConferenceRoom på sig er at når man booker høre der et rum til.

På price klassen ligger der en price attribut, det svare til den pris det koster for en person at opholde sig en dag på det tilhørende hotel.

Grunden til at der ikke er nogen Room tabel, men at Hotel i stedet har en Rooms attribut, er at man kan regne sig frem til hvor mange ledige rum der er på en bestemt dato ud fra Booking og Hotel. Dette gøres i den redundante BookingHelper Tabel.

Tabeldesign

Følgende checks er tilføjet til databasen:

Name på Hotel og CompanyCustomer er Unique fordi det ikke giver mening at have flere kunder eller hoteller der hedder det samme.

Datatypen for kundens rabat er sat til højest at kunne være 99.999% fordi jeg ikke syntes at det giver mening at gå højere.

Det er gjort umuligt at kunne have under 0 frie pladser i BookingHelper tabellen

Pga den tidligere omtalte redundans er der lavet en constraint på tabel niveau som sørger for at Booking og tilhørende ConferenceRoom høre til samme hotel. Den er lavet med en UDF ved navn booking_check_hotel_conferenceroom.

Se scriptet i Bilag 1.

2 Implantation af regler

Lukket i weekender

Ved at gøre så sql server ikke tillader at der oprettes bookinger i weekender kan man sikre at det aldrig kan forekomme i systemet.

For at finde ud af om en booking indeholder weekender skal man have spurgt sql server om der er en weekend i intervallet fra start datoen og det antal dage frem som bookingen vare.

Ved at implementere det i en constraint bliver kontrolleret hver gang uden undtagelser og at det er forholdsvis simpelt. Da der skal bruges to parametre, skal det være en table constraint og ikke bare en constraint på en enkelt attribut.

Igen bruges der en UDF som kaldes fra constrainten på Booking tabellen, se booking_check_no_weekend i Bilag 1

Ingen priser med tilbagevirkende kraft

Denne opgave tolkes som at der ikke må kunne oprettes, opdateres eller slettes priser før nuværende dato.

Valideringen skal ske både ved insert, update og delete men ikke på data der allerede ligger i databasen. Af den grund er det oplagt at bruge en trigger til at sikre denne regel.

Se triggeren no_new_prices_in_the_past_trigger i Bilag 1

3 Tre vigtige index

Price Tabellen

Der vil skulle slås priser op hver gang der skal afregnes. I denne tabel vil der ligge en række for hver dato og hotel kombination. Der vil altså hurtigt være mere end 100 rækker og dermed vil et nonclustered index have en god nok reduction factor til at blive brug. For at undgå lange opslags tider med stor datamængde oprettes følgende nonclustered index:

```
CREATE UNIQUE INDEX price_HotelId_ActiveFrom_Index ON Price (HotelId, ActiveFrom)
```

Indexet laves Unique for at gøre dobbeltoprettelser umulige.

ConferenceRoom Tabellen

I krav D2 skal der findes et ledigt lokale med plads til alle deltagere. Det antages at systemet har over 100 mødelokaler i alt. Derfor vil det give mening at oprettet følgende nonclustered index på ConferenceRoom tabellen:

```
CREATE INDEX ConferenceRoom_HotelId_Capacity_Index ON ConferenceRoom (HotelId, Capacity)
```

Booking Tabellen

I det fulde system vil der formentlig være brug for funktioner til at søge bookinger frem på de fleste kolonner i tabellen. I forbindelse med kundesupport kunne der fx være en funktion til at søge alle bookinger frem for en bestemt kunde. Det antages at der er over 100 kunder, så et nonclustered index vil være til gavn. Her er et index der understøtter denne funktion:

```
CREATE INDEX Booking_CompanyCustomerId_Index ON Booking (CompanyCustomerId)
```

4 Trigger til vedligeholdelse af BookingHelper tabellen

Se booking_helper_trigger i bilag 1

5 Stored Procedure til udregning af prisen for en booking

Det antages at prisen på den første dag gælder for hele bookingen og at kundens rabat skal medregnes i prisen.

Se booking_price_calculator_procedure i bilag 1

6 User defined function, det mindste ledige lokale til en booking

Se smallest_available_conference_room i bilag 1

7 Konsolapplikation, samtidighedskontrol og ADO

Koden der omtales i dette afsnit kan ses i bilag 2.

Min løsning – Pessimistisk Låsning

Den valgte løsning er lavet med pessimistisk låsning. Den største ulempe ved pessimistisk låsning er at transaktioner tage lang tid at gennemføre. Dette opstår fordi brugere kommer til at vente på hinanden.

Pessimistisk låsning laves ved at udlæse alt data der vises til brugeren eller bruges senere på samme transaktion som den dataændring der skal laves til sidst i programmet.

Alternativ løsning – Optimistisk låsning

Ved optimistisk låsning læses data ud og vises til brugeren på andre transaktioner end den der skal rette data sidst i programmet. Dermed låses de ikke og kan derfor ændres af andre brugere i tiden mellem visning og dataændring. På grund af at data kan være rettet kan det forekomme at man bliver nødt til at give en fejlbesked til brugeren om at data er ændret i stedet for at lade dataændringen gå igennem.

Sammenligning
Det to løsninger har som nævnt hver sine fordele og ulemper.

Pessimistisk låsning giver risiko for at brugere skal vente på hinanden, hvorimod optimistisk låsning giver risiko for at de kan lave tabt arbejde og få en tilhørende fejlbesked.

Valget af løsning er en afvejning mellem disse fordele og ulemper. I et produktionssystem med pessimistisk låsning ville man med fordel kunne bruge timeout i brugergrænsefladen så en bruger ikke kan låse de andre længere end fx 10 sekunder.

Valg af isolation level
Ved at vælge et passende isolation level kan man løse en række samtidighedsproblemer.

Dirty Read
Kan forekomme når en transaktion er ved at opdatere nogle records, men den vælger at rulle tilbage. Hvis en anden transaktion har læst de opdaterede records i mellemtiden, vil den få nogle data der ikke stemmer overens med resten af databasen.

Dirty read løses ved at bruge isolation level Read Commited. Det gør at den anden transaktion først få lov at læse, data som opdateres, efter den første har bekræftet sine opdateringer.

Lost Update
Kan forekomme når to transaktioner læser de samme records ind og forsøger at opdatere på baggrund af det læste data. Tidligere opdateringer vil altså gå tabt fordi de overskrives af den sidste transaktion.

Lost update løses ved at bruge isolation level Repeatable Read. Det gør at alt data der læses af transaktionen låses indtil den er afsluttet.

Phantom
Kan forekomme når en transaktion læser et antal rækker. Hvis en anden transaktion i mellemtiden indsætter en eller flere rækker som skulle have været med i den første, opstår problemet.

Phantom løses ved at bruge isolation level serializable. Det gør at alt data der ville komme med i en where statement som kunne få phantom rækker er låst.

Valg
Da funktionen smallest_available_conference_room læser i booking tabellen og vi efterfølgende indsætter i samme, vil der kunne opstå phantom rows. Derfor vil det bedste isolation level være serializable.

8 Konsolapplikation, deffered execution osv. med Linq To Entities
Se Bilag 3

Konklusion
Alle opgaver er nåede og sql delen er lavet uden brug af loops.

Bilag 1 - SQL Script

```
------------------------------
------------------------------
-- Databaser for udviklere   --
-- Eksamensopgave            --
-- Lars Søndergaard Petersen --
------------------------------
------------------------------

USE hotelbooking

--Dropping tables
DROP TABLE IF EXISTS Booking
DROP TABLE IF EXISTS BookingHelper
DROP TABLE IF EXISTS CompanyCustomer
DROP TABLE IF EXISTS Price
DROP TABLE IF EXISTS ConferenceRoom
DROP TABLE IF EXISTS Hotel

--Dropping UDFs
DROP FUNCTION IF EXISTS booking_check_hotel_conferenceroom
DROP FUNCTION IF EXISTS booking_check_no_weekend
DROP FUNCTION IF EXISTS smallest_available_conference_room

--Dropping stored procedures
DROP PROC IF EXISTS booking_price_calculator_procedure

--Dropping Triggers
DROP TRIGGER IF EXISTS no_new_prices_in_the_past_trigger

--Creating UDFs to be used in constraints
GO
CREATE FUNCTION booking_check_hotel_conferenceroom(@HotelId INT, @ConferenceRoomId INT)
RETURNS BIT
AS
BEGIN
IF (SELECT HotelId FROM ConferenceRoom WHERE Id = @ConferenceRoomId) = @HotelId
          RETURN 1
RETURN 0
END;
GO

CREATE FUNCTION booking_check_no_weekend(@StartDate DATE, @Days INT)
RETURNS BIT
AS
BEGIN
-
- Because of the constraint on bookings not being able to be more than 5 days we dont need to check fo
r that here.
-- søn = 1 -- man = 2 -- tir = 3 -- ons = 4-- tor = 5-- fre = 6-- lør = 7
--Checking if any of  the end dates are weekend days.
IF DATEPART(DW, @StartDate) = 1 OR DATEPART(DW, @StartDate) = 7
          RETURN 0
ELSE IF DATEPART(DW, DATEADD(DD, @Days-1, @StartDate)) = 1 OR DATEPART(DW, DATEADD(DD, @Days-
1, @StartDate)) = 7
          RETURN 0
-- Checking if we passed the weekend.
ELSE IF DATEPART(DW, @StartDate) > DATEPART(DW, DATEADD(DD, @Days-1, @StartDate))
          RETURN 0
RETURN 1
END;
GO

--Creating tables
CREATE TABLE CompanyCustomer
(
          Id INT IDENTITY PRIMARY KEY,
          Name VARCHAR(50) NOT NULL UNIQUE,
          Discount DECIMAL(5,3) NOT NULL,
)
```

```
CREATE TABLE Hotel
(
	Id INT IDENTITY PRIMARY KEY,
	Name VARCHAR(50) NOT NULL UNIQUE,
	Rooms INT NOT NULL,
)

CREATE TABLE ConferenceRoom
(
	Id INT IDENTITY PRIMARY KEY,
	HotelId INT NOT NULL FOREIGN KEY REFERENCES Hotel,
	Name VARCHAR(50) NOT NULL,
	Capacity INT NOT NULL,
)

CREATE TABLE BookingHelper
(
	HotelId INT NOT NULL FOREIGN KEY REFERENCES Hotel,
	FreeRooms INT NOT NULL CHECK(FreeRooms >= 0),
	Date DATE NOT NULL,
	PRIMARY KEY (HotelId, Date)
)

CREATE TABLE Price
(
	Id INT IDENTITY PRIMARY KEY,
	HotelId INT NOT NULL FOREIGN KEY REFERENCES Hotel,
	Price DECIMAL(18,3) NOT NULL,
	ActiveFrom DATE NOT NULL,
)

CREATE TABLE Booking
(
	Id INT IDENTITY PRIMARY KEY,
	HotelId INT NOT NULL FOREIGN KEY REFERENCES Hotel,
	ConferenceRoomId INT NOT NULL FOREIGN KEY REFERENCES ConferenceRoom ,
	CompanyCustomerId INT NOT NULL FOREIGN KEY REFERENCES CompanyCustomer,
	Participants INT NOT NULL,
	StartDate DATE NOT NULL,
	Days INT NOT NULL CHECK(Days IN (2,3,4,5)),

	CONSTRAINT booking_check_hotel_conferenceroom_constraint
	CHECK( 1 = dbo.booking_check_hotel_conferenceroom(HotelId, ConferenceRoomId)),

	CONSTRAINT booking_check_no_weekend_constraint
	CHECK( 1 = dbo.booking_check_no_weekend(StartDate, Days)),
)

--Inserting test data
INSERT INTO CompanyCustomer VALUES ('Elgiganten',12), ('Bilka',10), ('Coop',15), ('Ikea',30), ('Google
',50)
INSERT INTO Hotel VALUES ('Danhostel', 543), ('Guldsmeden',80), ('Zleep',90), ('Ritz',123), ('Comwell'
,230)
INSERT INTO ConferenceRoom (HotelId, Name, Capacity) VALUES (1,'Room D50',50 ), ( 1,'Room D25',25 ), (
 1,'Room D10',10 ), ( 1,'Room D5a',5 ), ( 1,'Room D5b',5 )
INSERT INTO ConferenceRoom (HotelId, Name, Capacity) VALUES (2,'Room G50',50 ), ( 2,'Room G25',25 ), (
 2,'Room G10',10 ), ( 2,'Room G5a',5 ), ( 2,'Room G5b',5 )
INSERT INTO ConferenceRoom (HotelId, Name, Capacity) VALUES (3,'Room Z50',50 ), ( 3,'Room Z25',25 ), (
 3,'Room Z10',10 ), ( 3,'Room Z5a',5 ), ( 3,'Room Z5b',5 )
INSERT INTO ConferenceRoom (HotelId, Name, Capacity) VALUES (4,'Room R50',50 ), ( 4,'Room R25',25 ), (
 4,'Room R10',10 ), ( 4,'Room R5a',5 ), ( 4,'Room R5b',5 )
INSERT INTO ConferenceRoom (HotelId, Name, Capacity) VALUES (5,'Room C50',50 ), ( 5,'Room C25',25 ), (
 5,'Room C10',10 ), ( 5,'Room C5a',5 ), ( 5,'Room C5b',5 )
INSERT INTO Price (HotelId, Price, ActiveFrom) VALUES (1,101,'2020-05-10'), (2,102,'2020-05-
10'), (3,103,'2020-05-10'), (4,104,'2020-05-10'), (5,105,'2020-05-10')
INSERT INTO Price (HotelId, Price, ActiveFrom) VALUES (1,201,'2020-05-25'), (2,202,'2020-05-
25'), (3,203,'2020-05-25'), (4,204,'2020-05-25'), (5,205,'2020-05-25')
INSERT INTO Booking (HotelId, ConferenceRoomId, CompanyCustomerId, Participants, StartDate, Days) VALU
ES (2, 6, 1, 45, '2020-05-18', 4), (2, 7, 1, 20, '2020-05-18', 3), (2, 8, 1, 9, '2020-05-18', 2)
INSERT INTO Booking (HotelId, ConferenceRoomId, CompanyCustomerId, Participants, StartDate, Days) VALU
ES (2, 6, 1, 45, '2020-05-27', 3), (2, 7, 1, 20, '2020-05-27', 3), (2, 8, 1, 9, '2020-05-27', 2)
INSERT INTO Booking (HotelId, ConferenceRoomId, CompanyCustomerId, Participants, StartDate, Days) VALU
ES (2, 6, 1, 45, '2020-06-10', 3), (2, 7, 1, 20, '2020-06-10', 3), (2, 8, 1, 9, '2020-06-10', 2)
```

```
INSERT INTO Booking (HotelId, ConferenceRoomId, CompanyCustomerId, Participants, StartDate, Days) VALU
ES (2, 6, 1, 45, '2020-06-17', 3), (2, 7, 1, 20, '2020-06-17', 3), (2, 8, 1, 9, '2020-06-17', 2)
INSERT INTO Booking (HotelId, ConferenceRoomId, CompanyCustomerId, Participants, StartDate, Days) VALU
ES (3, 12, 1, 45, '2020-05-27', 3), (3, 13, 1, 20, '2020-05-27', 3), (3, 14, 1, 9, '2020-05-27', 2)
INSERT INTO Booking (HotelId, ConferenceRoomId, CompanyCustomerId, Participants, StartDate, Days) VALU
ES (3, 12, 1, 45, '2020-06-10', 3), (3, 13, 1, 20, '2020-06-10', 3), (3, 14, 1, 9, '2020-06-10', 2)
INSERT INTO BookingHelper VALUES (2,6,'2020-05-18'), (2,6,'2020-05-19'), (2,26,'2020-05-
20'), (2,35,'2020-05-21')
GO

--Use to test the contraint on booking ( booking_check_hotel_conferenceroom )
--
INSERT INTO Booking (HotelId, ConferenceRoomId, CompanyCustomerId, Participants, StartDate, Days) VALU
ES (1, 6, 1, 45, '2020-05-20', 4)

--Use to test the contraint on booking ( booking_check_no_weekend )
--
INSERT INTO Booking (HotelId, ConferenceRoomId, CompanyCustomerId, Participants, StartDate, Days) VALU
ES (1, 6, 1, 45, '2020-05-25', 2)
--
INSERT INTO Booking (HotelId, ConferenceRoomId, CompanyCustomerId, Participants, StartDate, Days) VALU
ES (1, 6, 1, 45, '2020-05-23', 5)

--Trigger for making sure no prices are inserted for past data


GO
CREATE TRIGGER no_new_prices_in_the_past_trigger
ON Price
AFTER INSERT, UPDATE, DELETE
AS
IF EXISTS (SELECT * FROM inserted WHERE ActiveFrom < GETDATE())
BEGIN
          ROLLBACK TRAN
          RAISERROR('No price changes in the past.',16,1)
END

IF EXISTS (SELECT * FROM deleted WHERE ActiveFrom < GETDATE())
BEGIN
          ROLLBACK TRAN
          RAISERROR('Past prices can not be deleted.',16,1)
END
GO

--For testing price no_new_prices_in_the_past_trigger
--INSERT INTO Price (HotelId, Price, ActiveFrom) VALUES (1,200,'2029-05-25')
--INSERT INTO Price (HotelId, Price, ActiveFrom) VALUES (1,200,'2019-05-25')
--DELETE FROM Price
--UPDATE Price SET ActiveFrom = GETDATE()

--Creating the three most relevant indexes
CREATE UNIQUE INDEX Price_HotelId_ActiveFrom_Index ON Price (HotelId, ActiveFrom)
CREATE INDEX ConferenceRoom_HotelId_Capacity_Index ON ConferenceRoom (HotelId, Capacity)
CREATE INDEX Booking_CompanyCustomerId_Index ON Booking (CompanyCustomerId)
--Trigger for updating the redundant table BookingHelper, when a booking is inserted.
--This is made to only handle single inserts of bookings.
GO
```

```
CREATE TRIGGER booking_helper_trigger
ON Booking
AFTER INSERT
AS

DECLARE @HotelId INT
DECLARE @DaysToAdd INT
DECLARE @Participants INT
DECLARE @HotelRooms INT
DECLARE @StartDate DATE
DECLARE @EndDate DATE

SELECT @HotelId = HotelId FROM inserted
SELECT @DaysToAdd = (Days-1) FROM inserted
SELECT @Participants = Participants FROM inserted
SELECT @HotelRooms = Rooms FROM Hotel WHERE Id = @HotelId
SELECT @StartDate = StartDate FROM inserted
SELECT @EndDate =  DATEADD(DD, @DaysToAdd, @StartDate) FROM inserted; -- Semicolon needed before cte

--Inserting missing dates
--I went a bit overboard trying not to use a cursor here.
--Creating a cte containing the required dates.
WITH cte AS (
          SELECT
                    b.HotelId,
                    b.StartDate,
                    (SELECT Rooms FROM Hotel WHERE Id = b.HotelId) AS Rooms,
                    b.Days,
                    1 AS START
          FROM inserted b

          UNION ALL

          SELECT
                    c.HotelId,
                    DATEADD(DD, 1, c.StartDate),
                    (SELECT Rooms FROM Hotel WHERE Id = c.HotelId) AS Rooms,
                    c.Days,
                    START + 1

          FROM cte c
          WHERE START < c.Days
)

INSERT INTO BookingHelper (HotelId, FreeRooms, Date)
          --Excepting the cte with the existing dates to find the missing ones.
          --Adding the needed hotel rooms to only compare date and hotelid,
          --while prepping for insert if needed.
          SELECT HotelId, @HotelRooms AS FreeRooms, StartDate AS Date
          FROM cte

          EXCEPT

          SELECT HotelId, @HotelRooms AS FreeRooms, Date
          FROM BookingHelper
          WHERE HotelId = @HotelId AND (@StartDate <= Date AND @EndDate >= Date)

--Updating dates to decrease by the number of participants
--Fails when going below 0 becaus of the constraint on FreeRooms, thereby failing the booking insert.
UPDATE BookingHelper
SET FreeRooms = (FreeRooms - @Participants)
WHERE HotelId = @HotelId AND (@StartDate <= Date AND @EndDate >= Date)

GO
--For testing the BookingHelper trigger
--
INSERT INTO Booking (HotelId, ConferenceRoomId, CompanyCustomerId, Participants, StartDate, Days) VALU
ES (5, 21, 1, 10, '2020-05-18', 5)
```

```
--Creating stored proceadure to calculate the price of a booking with discount.
CREATE PROC booking_price_calculator_procedure (@BookingId INT, @BookingPrice DECIMAL(18,3) OUTPUT)
AS
          SELECT TOP 1 @BookingPrice =  b.Participants*b.Days*p.Price*((100-cc.Discount)/100)
          FROM Booking b
          INNER JOIN CompanyCustomer cc
          ON cc.Id = b.CompanyCustomerId
          INNER JOIN Price p
          ON p.HotelId = b.HotelId
          WHERE b.Id = @BookingId AND ActiveFrom >= b.StartDate
          ORDER BY ActiveFrom
GO

--For testing the store proceadure
--DECLARE @BookingId INT
--DECLARE @BookingPrice DECIMAL(18,3)
--SELECT @BookingId = 1
--EXEC booking_price_calculator_procedure  @BookingId, @BookingPrice OUTPUT
--PRINT @BookingPrice

GO
--Creating function to figure out what the smallest available
--conference room is for given parmeters ( parameters for making a booking ).
CREATE FUNCTION smallest_available_conference_room(@HotelId INT, @StartDate DATE, @Days INT, @Particip
ants INT)
RETURNS INT
AS
BEGIN
DECLARE @Result INT;

--Finding booked date-room combinations using CTE
WITH cte AS
(
          SELECT 1 AS START,b.Id, b.StartDate AS BookedDate, b.HotelId, ConferenceRoomId , Days
          FROM Booking b
          WHERE b.HotelId = @HotelId
          UNION ALL
          SELECT START + 1,c.Id, DATEADD(DD, 1, c.BookedDate) AS BookedDate , c.HotelId, ConferenceRoo
mId , Days
          FROM cte c
          WHERE START < c.Days AND HotelId = @HotelId
)

--Selcting the lowest capacity room that fits the requirements
SELECT TOP (1) @Result = Id FROM ConferenceRoom
WHERE Id NOT IN
(
          --Selecting rooms within the duration that can't be used
          SELECT DISTINCT ConferenceRoomId FROM cte
          --This where could potentially be moved into the cte to improve performance and lock less.
          WHERE BookedDate >= @StartDate AND BookedDate <= DATEADD(DD, @Days-1, @StartDate)
) AND
Capacity >= @Participants AND
Hotelid = @HotelId
ORDER BY Capacity ASC

RETURN @Result
END
GO

-- For testing the UDF smallest_available_conference_room
--DECLARE @Result INT
--EXEC @Result = smallest_available_conference_room 2, '2020-05-18', 3, 30
--SELECT @Result
```

Bilag 2 - Konsolapplikation ADO.Net

```
//Instance variables
private int _hotelId;
private int _conferenceRoomId;
private SqlConnection _connection;
private SqlTransaction _transaction;

//User input
private int _customerId;
private string _hotelName;
private int _participants;
private DateTime _startDate;
private int _days;

//Method used for testing
private void GetInputFromMethod()
{
    _customerId = 1;
    //_hotelName = "Danhostel";
    _hotelName = "Guldsmeden";

    _participants = 4;
    _startDate = DateTime.Now.AddDays(4);
    _days = 5;
}

public void Run()
{
    try
    {
        Console.WriteLine("Med denne applikation kan du booke et hotel arrangement.");

        //GetInputFromUser();

        //For testing the code without entering data every time.
        GetInputFromMethod();

        SetHotelIdFromName();
        _transaction = GetConnection().BeginTransaction(IsolationLevel.Serializable);

        //Find the smallest vacant room
        var vacantRoom = GetVacantRoom();
        _conferenceRoomId = vacantRoom;

        //My first intuition was to create the same logic contained in booking_helper_trigger.
        //Inserting the booking and checking if that went well, using the trigger seems smarter.

        try
        {
            InsertBooking();
        }
        catch (Exception)
        {
            throw new Exception("Der er desværre ikke plads på hotelet i den valgte periode.");
        }

        //Find name of the room.
        var roomName = GetNameOfRoom(vacantRoom);

        //Asking if the user wants to confirm the booking.
        Console.WriteLine("Vil du genenmføre bookingen, og booke rummet: {0} ?", roomName);
        Console.WriteLine("Tryk J for Ja eller N for Nej");
        if (ShouldProceedWithBooking())
        {
            _transaction.Commit();
            Console.WriteLine("Bookingen er gennemført.");
        }
        else
        {
            _transaction.Rollback();
            Console.WriteLine("Bookingen er annulleret.");
        }
    }
```

```
    catch (Exception ex)
    {
        if (_transaction != null)
        {
            _transaction.Rollback();
        }
        Console.WriteLine(ex.Message);
    }
    finally
    {
        if (_connection != null)
        {
            _connection.Close();
        }
        Console.WriteLine("Tryk på en vilkårlig tast for at afslutte.");
        Console.ReadKey();
    }
}

private void InsertBooking()
{
    var query = "INSERT INTO Booking (HotelId, ConferenceRoomId, CompanyCustomerId, Participants, Star
tDate, Days) " +
                                "VALUES (@HotelId, @ConferenceRoomId, @CompanyCustomerId, @Participant
s, @StartDate, @Days) ";

    SqlCommand command = new SqlCommand(query, GetConnection());
    command.Transaction = _transaction;
    command.Parameters.AddWithValue("@HotelId", _hotelId);
    command.Parameters.AddWithValue("@ConferenceRoomId", _conferenceRoomId);
    command.Parameters.AddWithValue("@CompanyCustomerId", _customerId);
    command.Parameters.AddWithValue("@Participants", _participants);
    command.Parameters.AddWithValue("@StartDate", _startDate.Date);
    command.Parameters.AddWithValue("@Days", _days);
    command.ExecuteNonQuery();
}

private string GetNameOfRoom(int vacantRoom)
{
    var query = "SELECT Name " +
                "FROM ConferenceRoom " +
                "WHERE Id = @RoomId ";

    SqlCommand command = new SqlCommand(query, GetConnection());
    command.Transaction = _transaction;
    command.Parameters.AddWithValue("@RoomId", vacantRoom);
    SqlDataReader reader = command.ExecuteReader();

    string result = "";
    using (reader)
    {
        if (reader.HasRows)
        {
            while (reader.Read())
            {
                result = reader.GetString(0);
            }
        }
        else
        {
            throw new Exception("Der blev ikke fundet et mødelokale med det id.");
        }
    }
    return result;
}
```

```
private void SetHotelIdFromName()
{
    var query = "SELECT Id " +
                "FROM Hotel " +
                "WHERE Name = @HotelName ";

    SqlCommand command = new SqlCommand(query, GetConnection());
    command.Parameters.AddWithValue("@HotelName", _hotelName);
    SqlDataReader reader = command.ExecuteReader();

    using (reader)
    {
        if (reader.HasRows)
        {
            while (reader.Read())
            {
                _hotelId = reader.GetInt32(0);
            }
        }
        else
        {
            throw new Exception("Der findes desværre ikke et hotel med det navn.");
        }
    }
}

private int GetVacantRoom()
{
    var query = "DECLARE @Result INT " +
                "EXEC @Result = smallest_available_conference_room @HotelId, @StartDate, @Days, @Parti
cipants " +
                "SELECT @Result ";

    SqlCommand command = new SqlCommand(query, GetConnection());
    command.Transaction = _transaction;
    command.Parameters.AddWithValue("@HotelId", _hotelId);
    command.Parameters.AddWithValue("@StartDate", _startDate.Date);
    command.Parameters.AddWithValue("@Days", _days);
    command.Parameters.AddWithValue("@Participants", _participants);

    SqlDataReader reader = command.ExecuteReader();
    int result = 0;
    using (reader)
    {
        if (reader.HasRows)
        {
            while (reader.Read())
            {
                if (reader.IsDBNull(0))
                {
                    throw new Exception("Der blev desværre ikke fundet et passende ledigt mødelokale."
);
                }
                return reader.GetInt32(0);
            }
        }
        else
        {
            throw new Exception("Der blev desværre ikke fundet et passende ledigt mødelokale.");
        }
    }
    return result;
}
```

```
private void GetInputFromUser()
{

    Console.WriteLine("Indtast kunde id:");
    _customerId = GetPositiveIntFromUser();

    Console.WriteLine("Indtast hotel navn:");
    _hotelName = Console.ReadLine();

    Console.WriteLine("Indtast antal deltagere:");
    _participants = GetPositiveIntFromUser();

    Console.WriteLine("Indtast start dato:");
    _startDate = GetFutureDateFromUser();

    Console.WriteLine("Indtast arrangementets varighed i dage:");
    _days = GetPositiveIntFromUser();
}

private void SleepFor(int pauseTimeInSeconds)
{
    for (int i = pauseTimeInSeconds; i > 0; i--)
    {
        Console.WriteLine(i);
        Thread.Sleep(1000);
    }
}

private SqlConnection GetConnection()
{
    string connectionString = "Server=localhost,1439;Database=hotelbooking;User=sa;Password=MyPass@wor
d;";

    if (_connection == null)
    {
        _connection = new SqlConnection(connectionString);
        _connection.Open();
    }
    return _connection;
}

private int GetPositiveIntFromUser()
{
    while (true)
    {
        int input;
        if (int.TryParse(Console.ReadLine(), out input) && input > 0)
        {
            return input;
        }
        Console.WriteLine("Indtast venligst et positivt heltal mindre end: " + int.MaxValue);
    }
}

private DateTime GetFutureDateFromUser()
{
    Console.WriteLine("Datoformat: DD-MM-YYYY");
    while (true)
    {
        DateTime result;
        var userInput = Console.ReadLine();
        if (userInput.Length == 10 && DateTime.TryParse(userInput, out result))
        {
            if (result.Date >= DateTime.Now.Date)
            {
                return result;
            }
            Console.WriteLine("Det er ikke tilladt at booke i fortiden.");
        }
        Console.WriteLine("Indtast venligst en dato i det korrekte format:");
    }
}
```

```
private bool ShouldProceedWithBooking()
{
    var valg = Console.ReadKey(true).KeyChar;
    var positiveValg = new char[] { 'J', 'j' };
    var negativeValg = new char[] { 'N', 'n' };
    while (!positiveValg.Contains(valg) && !negativeValg.Contains(valg))
    {
        Console.WriteLine("Tryk J for Ja eller N for Nej");
        valg = Console.ReadKey(true).KeyChar;
    }
    return positiveValg.Contains(valg);
}
```

Bilag 3 - Konsolapplikation Linq To Entities

```
private static hotelbookingEntities db = new hotelbookingEntities();
private const string seperator = "------------------------------------------------------------------------
------";
private const string functionSeperator = "############################################################
##############";
static void Main(string[] args)
{
    //Showing generated sql output in console
    //db.Database.Log = Console.WriteLine;

    ConferenceRoomOverviewDefferedLazy();
    ConferenceRoomOverviewImmidiateEager();
    CreateNewConfConferenceRoom();
    HotelWorkDayCalculator();

    db.Dispose();
    Console.WriteLine("Tryk på en tast for at afslutte.");
    Console.ReadKey();
}

private static void ConferenceRoomOverviewDefferedLazy()
{
    Console.WriteLine(functionSeperator);
    Console.WriteLine("Room overview, enter room name:");
    var roomName = "Room G50";
    // Remove input for easy testing
    roomName = Console.ReadLine();

    var room = db.ConferenceRooms.FirstOrDefault(r => r.Name == roomName);
    if (room != null)
    {
        var result = room.Bookings.Where(b => b.StartDate > DateTime.Now).OrderBy(b => b.StartDate);
        if (result.Any())
        {
            string allignementString = "{0,-15}{1,-15}{2,-10}{3,-10}";

            Console.WriteLine(seperator);
            Console.WriteLine(allignementString, "Navn", "Fradato", "Varighed", "Antal deltagere");
            Console.WriteLine(seperator);
            foreach (var booking in result)
            {
                Console.WriteLine(allignementString, booking.CompanyCustomer.Name, booking.StartDate.T
oShortDateString(), booking.Days, booking.Participants);
            }
            Console.WriteLine(seperator);
            Console.WriteLine();
        }
        else
        {
            Console.WriteLine("Der blev ikke fundet nogle fremtidige bookinger for det mødelokale");
        }
    }
    else
    {
        Console.WriteLine("Mødelokalet blev ikke fundet..");
    }
    Console.WriteLine(functionSeperator);
}
```

```
private static void ConferenceRoomOverviewImmidiateEager()
{
    Console.WriteLine(functionSeperator);
    Console.WriteLine("Room overview, enter room name:");
    var roomName = "Room G50";
    // Remove input for easy testing
    roomName = Console.ReadLine();
    var room = db.Bookings.Include(b => b.CompanyCustomer).Where(b => b.StartDate > DateTime.Now && b.
ConferenceRoom.Name == roomName).OrderBy(b => b.StartDate).ToList();
    if (room != null)
    {
        var result = room;
        if (result.Any())
        {
            string allignementString = "{0,-15}{1,-15}{2,-10}{3,-10}";

            Console.WriteLine(seperator);
            Console.WriteLine(allignementString, "Navn", "Fradato", "Varighed", "Antal deltagere");
            Console.WriteLine(seperator);
            foreach (var booking in result)
            {
                Console.WriteLine(allignementString, booking.CompanyCustomer.Name, booking.StartDate.T
oShortDateString(), booking.Days, booking.Participants);
            }
            Console.WriteLine(seperator);
            Console.WriteLine();
        }
        else
        {
            Console.WriteLine("Der blev ikke fundet nogle fremtidige bookinger for det mødelokale");
        }
    }
    else
    {
        Console.WriteLine("Mødelokalet blev ikke fundet..");
    }
    Console.WriteLine(functionSeperator);
}

private static void CreateNewConfConferenceRoom()
{
    Console.WriteLine(functionSeperator);
    Console.WriteLine("Oprettelse af nyt mødelokale.");
    Console.WriteLine("Tast hotel id:");
    var hotelId = GetPositiveIntFromUser();
    Console.WriteLine("Tast lokale navn:");
    var roomName = Console.ReadLine();
    Console.WriteLine("Indtast kapacitet:");
    var capacity = GetPositiveIntFromUser();

    db.ConferenceRooms.Add(new ConferenceRoom()
    {
        HotelId = hotelId,
        Capacity = capacity,
        Name = roomName
    });
    db.SaveChanges();
    Console.WriteLine(functionSeperator);
}
```

```
private static void HotelWorkDayCalculator()
{
    Console.WriteLine(functionSeperator);
    Console.WriteLine("Udregning af hvad der sker på et hotel en bestemt dato.");
    Console.WriteLine("Indtast hotel id:");

    var hotelId = 2;
    // Remove input for easy testing
    hotelId = GetPositiveIntFromUser();

    Console.WriteLine("Indtast dato:");
    var date = DateTime.Parse("19-05-2020");
    // Remove input for easy testing
    date = GetDateFromUser();

    var activeBookings = db.Bookings.Where(b => b.HotelId == hotelId && (b.StartDate <= date && DbFunc
tions.AddDays(b.StartDate, b.Days) > date)).ToList();
    int breakfastAndLunch = 0;
    int dinnerAndNights = 0;
    foreach (var booking in activeBookings)
    {
        breakfastAndLunch += booking.Participants;

        if (booking.StartDate.AddDays(booking.Days - 1) != date)
        {
            dinnerAndNights += booking.Participants;
        }
    }
    if (activeBookings.Any())
    {
        Console.WriteLine(seperator);
        Console.WriteLine("Morgenmad:\t" + breakfastAndLunch);
        Console.WriteLine("Frokost:\t" + breakfastAndLunch);
        Console.WriteLine("Aftensmad:\t" + dinnerAndNights);
        Console.WriteLine("Overnatninger:\t" + dinnerAndNights);
        Console.WriteLine(seperator);

    }
    else
    {
        Console.WriteLine("No bookings on that date.");
    }
    Console.WriteLine(functionSeperator);
}

private static int GetPositiveIntFromUser()
{
    while (true)
    {
        int input;
        if (int.TryParse(Console.ReadLine(), out input) && input > 0)
        {
            return input;
        }
        Console.WriteLine("Indtast venligst et positivt heltal mindre end: " + int.MaxValue);
    }
}

private static DateTime GetDateFromUser()
{
    Console.WriteLine("Datoformat: DD-MM-YYYY");
    while (true)
    {
        DateTime result;
        var userInput = Console.ReadLine();
        if (userInput.Length == 10 && DateTime.TryParse(userInput, out result))
        {
            return result;
        }
        Console.WriteLine("Indtast venligst en dato i det korrekte format:");
    }
}
```