

PART 2

Implementation of Happy Vally Campsite Database



JOE SHEPHERD

Contents

Table Creation and Inserting Data	2
Guest	2
Party	4
Party Members	5
Vehicle	6
Accommodation	7
Booking	8
Invoice	10
Booking Status	11
Activity	12
Activity Booking	13
Activity Organiser	14
Activity Invoice	16
Banned Guest	17
Scenarios and Queries	19
1	19
2	19
3	21
4	22
5	23
6	24
7	24
8	25
9	25
10	26
11	26
Triggers/ Stored Procedures and Functions	28
A	28
В	30
C	21

Table Creation and Inserting Data

For the implementation of the Robinsons family campsite database, the first step was to create the tables on SQL Server. Having looked at the feedback that was given to me it was clear that there were some errors in my original design. So before creating the tables, I made some changes to the entity relation diagram from part 1. All the tables were created using SQL statements which I have given screenshots for each table below.

Data Type

int

varchar(4)

varchar(20)

varchar(20)

varchar(100)

varchar(10)

varchar(20)

varchar(15)

varchar(20)

varchar(80)

varchar(3)

date

Guest

Figure 1

```
CREATE TABLE Guest (
                                                                                        Column Name
                  INT PRIMARY KEY IDENTITY (1,1),
 GuestID
                 VARCHAR(4)
                     CONSTRAINT chkTitle
                     CHECK (Title IN ('Mr', 'Mrs', 'Miss', 'Ms', 'Dr')),
                                                                                 Title
                 VARCHAR(20) NOT NULL,
 LastName
                 VARCHAR(20) NOT NULL,
                                                                                 FirstName
 DOB
                 DATE
                               NOT NUL
                     CONSTRAINT chkBrithDate
                                                                                 LastName
                     CHECK (DOB >= '1916-01-01'),
 StreetAddress
                 VARCHAR(100) NOT NULL,
                                                                                 DOB
                 VARCHAR(10) NOT NULL,
 PostCode
                                                                                 StreetAddress
 City
                 VARCHAR(20)
                              NOT NULL.
                 VARCHAR(15) NOT NULL,
 Country
                                                                                 PostCode
                 VARCHAR(20) NOT NULL,
VARCHAR(80) NOT NULL UNIQUE
 Email
                                                                                 City
                     CONSTRAINT chkEmail
                     CHECK (Email LIKE '%_@
                                                                                 Country
                 VARCHAR(3)
 LeadGuest
                               NOT NULL
                     CONSTRAINT leadGuest
                                                                                 PhoneNo
                     CHECK (LeadGuest IN('YES','NO'))
                                                                                 Email
                                                                                 LeadGuest
Messages
Command(s) completed successfully.
```

Figure 1 shows the Guest table. I added a leadguest column from my design in part 1 to show who is a lead guest. It has a check constraint to allow only values of 'yes' or 'no'. This table has three other check constraints to improve data integrity. chkTitle only allows values of a person's title to be inserted in the title column. chkBirthDate checks that people do not enter unrealistic values of a person's birth date as I figured no one would be over 100 years old. Finally, chkEmail ensures that the format of the data inserted into this column is the same format as an email, so it only can contain email addresses. At first I set this as unique, however later removed this constraint as it caused problems later in my design.

For the primary keys that contain IDs, an identity constraint is added so that the values are automatically inserted as a unique number starting from 1.

```
Figure 1.1
```

```
3.sql - J...Voe Shepherd (54))* → X

□ALTER TABLE Guest

ADD CONSTRAINT LeadGuestDefault

DEFAULT 'NO' FOR LeadGuest;

| ALTER COLUMN LeadGuest VARCHAR(3) NULL;
```

I also later changed the leadguest column (figure 1.1) to allow NULL values, and setting the default value as NO, so that data in this column only needs to be entered as 'YES' for lead guests which would make it simpler to insert data.

Figure 1.2

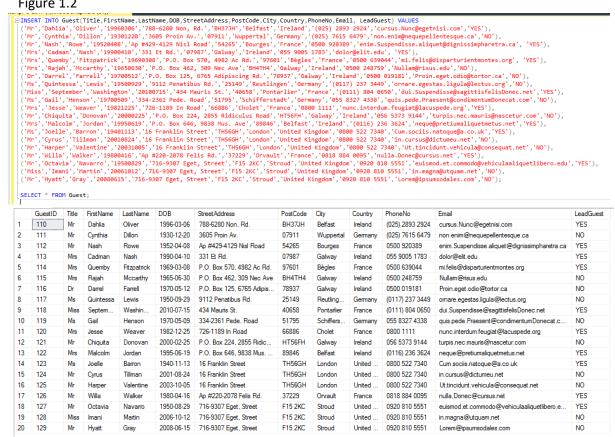
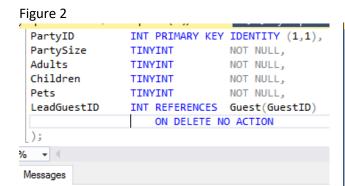


Figure 1.3

```
INSERT INTO Guest(Title,FirstName,LastName,DOB,StreetAddress,PostCode,City,Country,PhoneNo,Email, LeadGuest)
 VALUES ('mr|','Joe','David','19100306','788-6280 Non, Rd.','BH37JH','Belfast','Ireland','(025) 2893 2924','cursus.Nuncegetnisi.com','Y');
Messages
Msg 547, Level 16, State 0, Line 1
The INSERT statement conflicted with the CHECK constraint "chkBrithDate". The conflict occurred in database "Happy Vally Campsite", table "db
The statement has been terminated.
   INSERT INTO Guest(Title,FirstName,LastName,DOB,StreetAddress,PostCode,City,Country,PhoneNo,Email, LeadGuest)
   VALUES ('mrr','Joe','David|','19100306','788-6280 Non, Rd.','BH37JH','Belfast','Ireland','(025) 2893 2924','cursus.Nuncegetnisi.com','Y');
0% - 4
 Msg 547, Level 16, State 0, Line 1
  The INSERT statement conflicted with the CHECK constraint "chkTitle". The conflict occurred in database "Happy Vally Campsite", table "dbo.Gu
  The statement has been terminated.
  □INSERT INTO Guest(Title,FirstName,LastName,DOB,StreetAddress,PostCode,City,Country,PhoneNo,Email, LeadGuest)
   VALUES ('mr','Joe','David','1997|0306','788-6280 Non, Rd.','BH37JH','Belfast','Ireland','(025) 2893 2924','cursus.Nuncegetnisi.com','Y');
Messages
  Msg 547, Level 16, State 0, Line 1
  The INSERT statement conflicted with the CHECK constraint "chkEmail". The conflict occurred in database "Happy Vally Campsite", table "dbo.Gu
```

Figure 1.2 is the code to insert the values into the Guest table, with the results of the SQL statement shown underneath. To test that all the check constraints are working correctly, I entered incorrect data into the columns to see the results (see figure 1.3).

Party



Command(s) completed successfully.

could ruin data integrity in the database.

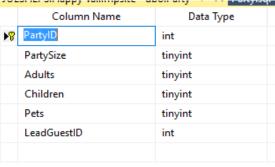
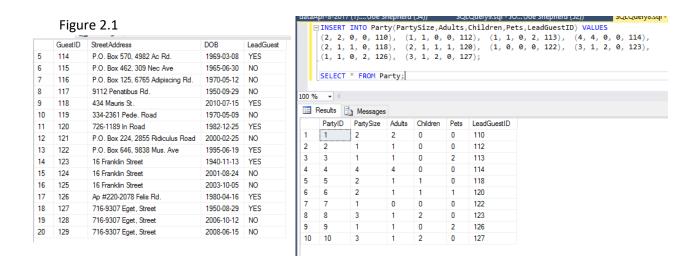


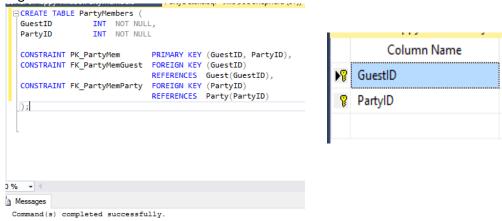
Figure 2 shows the party table. LeadGuest is a foreign key relating to the Guest table, with an on delete rule set as no action so that if the partyID is references in another table, it cannot be deleted as it



To insert data in the party column, for simplicity I used a select statement to return all information to the guest table, and entered the leadguest ID by looking at the table (see figure 2.1)

PartyMembers

Figure 3



This is the PartyMembers table (Figure 3). Originally I named this party details in part 1, however the way I designed the table it would only allow for one guest for each party. To overcome this I made changes to the guest table as seen previously, and completely re-designed this table. GuestID and PartyID and both foreign keys and primary keys. It shows which guest is associated with which party.

Data Type

int

int

I did not set any constraints for the on update or on delete, as sql server sets them as NO ACTION automatically which is what I would have set them as.

Figure 3.1

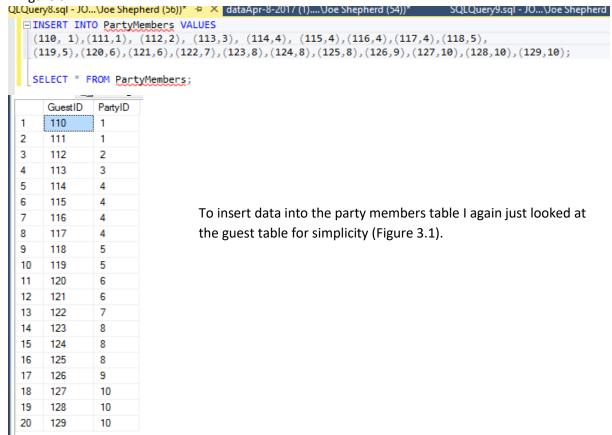
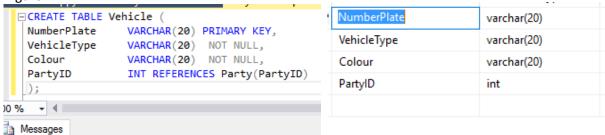


Figure 4



Command(s) completed successfully.

Figure 4.1

```
SQLQuery8.sql - JO...\Joe Shepherd (50))* -P X dataApr-8-2017 (1)....\Joe Shepherd (54))*

| INSERT INTO Vehicle( NumberPlate, VehicleType, Colour, PartyID) VALUES ('00H 848Y', 'Car', 'Black', 1), ('BP99 NPC', 'Motorbike', 'White', 2), ('LB56 VWD', 'Motorbike', 'Grey', 3), ('NA57 ZGG', 'Car', 'Blue', 4), ('KH55 HLJ', 'Car', 'Silver', 5), ('ZT66 YHT', 'Car', 'White', 6), ('FG53 YTT', 'Car', 'Orange', 7), ('JK43 HDG', 'Car', 'Green', 8), ('HH00 5HR', 'Car', 'Yellow', 9), ('FS33 GGH', 'Car', 'Black', 10);

| SELECT * FROM Vehicle;
```

	NumberPlate	VehicleType	Colour	PartyID
1	BP99 NPC	Motorbike	White	2
2	FG53 YTT	Car	Orange	7
3	FS33 GGH	Car	Black	10
4	HH00 5HR	Car	Yellow	9
5	JK43 HDG	Car	Green	8
6	KH55 HLJ	Car	Silver	5
7	LB56 VWD	Motorbike	Grey	3
8	NA57 ZGG	Car	Blue	4
9	OOH 848Y	Car	Black	1
10	ZT66 YHT	Car	White	6

The vehicle table (figure 4) has a foreign key that references the party table. I could have added a check constraint that would check to see if the format of the data entered in the number plate column is that of a number plate, however I was unsure of how to do this. Figure 4.1 shows the insert statements and the results from the inputted values. To insert the data, I just picked each partyID from the party table.

Accommodation

Figure 5



The accommodation table (figure 5) only has one check constraint, named chkType. What this constraint does is checks to see that the data inserted is either 'Tent', 'Motorhome', 'Caravan' or 'Glamping yurt'. This is to improve data integrity so there are not any random values inserted into accommodation type, as the campsite only provides these four types of accommodation.

Figure 5.1

```
SQLQuery8.sql - JU...Voe Shepherd (50))  

SQLQuery8.sql - JU...Voe Shepherd (50)]  

SQLQuery8.sql - Ju...Voe
```

To insert into the accommodation table (figure 5.1) the ID column is not required as it is set as Identity, therefore inserts a unique number automatically. Underneath the insert statement is the code to try and insert an incorrect value for the accType column. The screenshot show that the constraint on this column works correctly.

	AccomodationID	AccType	Capacity	PricePerNight
1	25	Caravan	8	50.00
2	26	Tent	4	15.00
3	27	Motorhome	4	30.00
4	28	Glamping Yurt	5	80.00
5	29	Tent	6	22.00
6	30	Tent	2	10.00
7	31	Caravan	6	35.00
8	32	Motorhome	7	35.00
9	33	Glamping Yurt	5	80.00
10	34	Glamping Yurt	6	85.00
11	35	Tent	5	18.00
12	36	Caravan	7	43.00

Booking

Figure 6

```
□CREATE TABLE Booking (
                                                                       Column Name
                                                                                               Data Type
  BookingID
                    INT PRIMARY KEY IDENTITY(1,1),
                    INT REFERENCES Party(PartyID),
                                                                  BookingID
                                                                                          int
  PartyID
   AccommodationID
                    INT REFERENCES Accommodation(AccommodationID),
                                                                  PartyID
                                                                                          int
  CheckIn
                    DATE
                          NOT NULL
                    CONSTRAINT chkDate
                                                                  AccommodationID
                                                                                          int
                    CHECK
                             (CheckIn >= GETDATE()),
                           NOT NULL,
  CheckOut
                    DATE
                                                                  CheckIn
  );
                                                                  CheckOut
                                                                                          date
h Messages
 Command(s) completed successfully.
ALTER TABLE Booking ADD
  CONSTRAINT chkDate CHECK ( CheckIn !< (GETDATE() - 1));</pre>
  CONSTRAINT chkCheckOutDate CHECK (CheckOut > CheckIn);
                                             accommodation.squ...soc onep
                          abo.booking

□ ALTER TABLE Booking

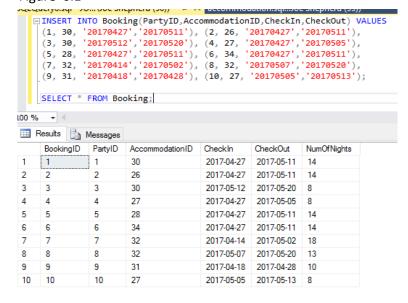
    ADD NumOfNights AS (DATEDIFF(day, CheckIn, CheckOut));
```

The Booking table (figure 6) has one check constraint at the table level. When creating the table I realised the constraint chkDate was not allowing me to insert a check in date that was the current date, therefore I had to use the ALTER TABLE statement to change this constraint. It checks that the check in date is not less than the current date -1, because if I had just used GETDATE(), only values greater than the current date could be entered. It also checks that the checkout date is in the future from the check in date, as if you could set a date in the past this would not be realistic data.

I later added another column named NumOfNights to the table to make some of the sql statements in the invoice table easier to execute. This column is a computed column that works out how many nights a guest is staying. I used the DATEDIFF function which takes two parameters of DATE and works out the difference.

Figure 6.1 shows the insert statement and the table after the data had been entered.

Figure 6.1



To ensure that accommodation cannot get double booked, I added a trigger onto the booking table (figure 6.2). This trigger is activated after data is inserted or updated and checks to see if there is two of the same accommodation IDs that have an overlapping date in either the check in or checkout date. If a row exists it means that accommodation has been double booked, then the database is rolled back to before the data was entered, so the booking does not go through and an error appears.

Figure 6.2

```
CK CONTRSINT...oe Shepherd (54))* + X SQLQuery1.sql - not connected*
 1 GCREATE TRIGGER Booked ON dbo.Booking
     after INSERT, UPDATE
 4
 5
   8
        SELECT 1 FROM inserted as x
 9
     INNER JOIN Booking ON x.AccommodationID = Booking.AccommodationID AND x.BookingID != Booking.BookingID
10
11
      WHERE x.CheckOut >= Booking.CheckIn AND Booking.CheckOut >= x.CheckIn
12
13
14
15 BEGIN
16
     ROLLBACK TRANSACTION;
17
      RETURN
18
     END
```

Figure 6.3

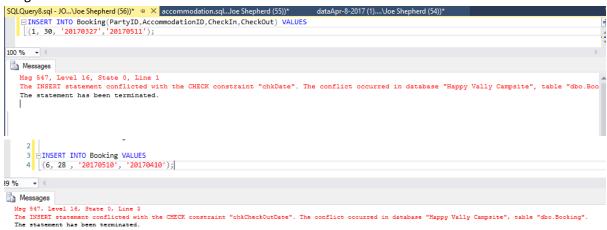
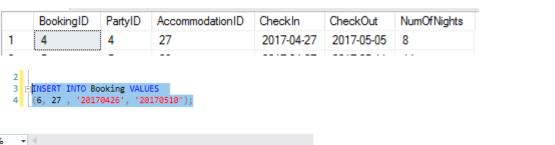


Figure 6.4



```
% 

Messages

Msg 3609, Level 16, State 1, Line 3

The transaction ended in the trigger. The batch has been aborted.
```

To ensure the check constraints on the booking table did the correct thing, I inserted some test data (figure 6.3). The first insert statement tests that a date in the past cannot be set for the checkin date. The second insert statement checks that a checkout date set before the checkin date cannot be inserted in the table. Both came back with errors which shows both constraints are working correctly.

To test that the trigger does not allow for double bookings, I selected a row from the booking table (figure 6.4), and attempted to enter a new booking with the same accommodation ID and a date that overlaps with the other booking date. As you can see in the screenshot the error appears as 'transaction ended in trigger' therefore accommodation cannot be overbooked.

Invoice



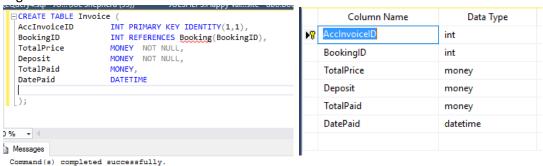


Figure 7.1



OllopiosID	PoolsinalD	TotalDrico	Donosit	TotalPaid
ccinvoiceid	BOOKINGID	TotalPfice	Deposit	rotainaid
61	1	140.00	35.00	35.00
52	2	210.00	52.50	52.50
63	3	80.00	20.00	20.00
64	4	240.00	60.00	60.00
65	5	1120.00	280.00	280.00
66	6	1190.00	297.50	297.50
67	7	630.00	157.50	630.00
88	8	455.00	113.75	113.75
69	9	350.00	87.50	350.00
35	25	150.00	0.00	150.00
	52 53 54 55 56 67 58 59	61 1 52 2 63 3 54 4 65 5 66 6 67 7 58 8 69 9	31 1 140.00 32 2 210.00 33 3 80.00 34 4 240.00 35 5 1120.00 36 6 1190.00 37 7 630.00 38 8 455.00 39 9 350.00	1 140.00 35.00 52 2 210.00 52.50 63 3 80.00 20.00 64 4 240.00 60.00 65 5 1120.00 280.00 66 6 1190.00 297.50 67 7 630.00 157.50 68 8 455.00 113.75 69 9 350.00 87.50

Figure 7 shows the sql statement to create the invoice table. I did not set any constraints for this table. To insert the data into this table (figure 7.1) I used the WHILE statement to automatically enter all the bookings into the invoice table. By declaring a variable INT that iterates through 1 to 25 as 25 was the

max bookingID at the time, then the bookingID value inserted was that of the variable, and I used a subquery with the SUM function to work out how much the total cost of each booking is from the NumofNights column in the booking table and PricePernight column from the accommodation table. After that I used the update statement to work out the deposit by multiplying the total cost by 0.25 (25%) only if the check in date was not the current date because guests who are making a booking the same day as the check in they must pay all in advance. The total paid was set as the deposit amount as the deposit must be payed straight away, unless the deposit was 0 which the total paid was the full amount. I used this method as it was a lot easier than inserting all the data manually and so that the data entered was correct.

Booking Status



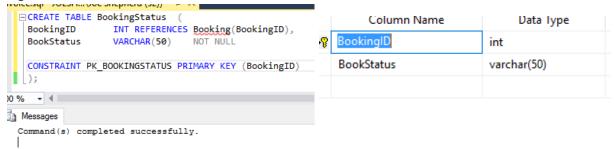


Figure 8.1

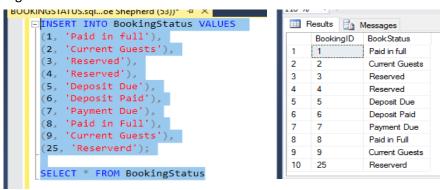
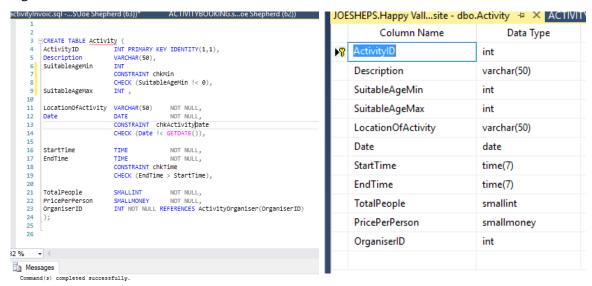


Figure 8 shows the create table statement for the booking status table. 8.1 shows the insert statement and the table after data has been inserted. This table has no check constraints.

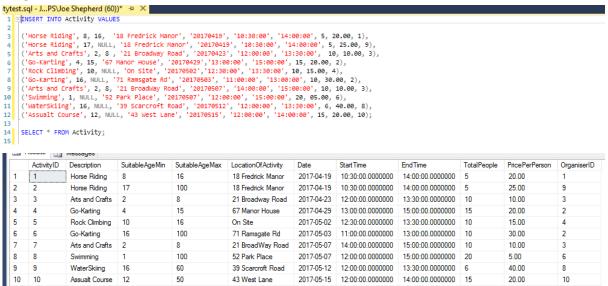
Activity

Figure 9



This is the activity table (figure 9). I added three check constraints. chkMin ensures that the suitable minimum age is not less than 0. chkActivityDate ensures that the date of the activity is not set in the past, and chkTime ensures that the end time is not before the starting time. When inserting data into the table I left some max age values as NULL. I thought there was not much need of setting a max age if an activity was to allow all ages above a certain age. However I couldn't get some queries to work so later updates the values (figure 9.1).

Figure 9.1



I inserted a range of test data to see if the check constraints do not allow data that is not supposed to be allowed (figure 9.2) As the screenshots show all three constraints are working correctly.

Figure 9.2

```
SQLQuery3.sql -JO...Voe Shepherd (56))* ** SQLQuery2.sql -JO...Voe Shepherd (54))

1 chicker Intro Activity WallES

2 [ "Throne Riding", 1, 60, "anywhere", "20170505", "10:30:00", "12:00:00", 5, 20.00, 1);

89 % - 4

Messages

Mag 897, Level 16, State 0, Line 1

The INSERT INTO Activity ValueS

2 [ "Throne Riding", 10, 60, "anywhere", "20170505", "10:30:00", "12:00:00", 5, 20.00, 1);

SQLQuery3.sql -JO...Voe Shepherd (56))* ** SQLQuery2.sql -JO...Voe Shepherd (55))

SQLQuery1.sql -JO...Voe Shepherd (56))* ** SQLQuery2.sql -JO...Voe Shepherd (55))

SQLQuery1.sql -JO...Voe Shepherd (54))

1 CHICKET INTO Activity ValueS

2 [ "Throne Riding", 10, 60, "anywhere", "20170365", "10:30:00", "12:00:00", 5, 20.00, 1);

89 % - 4

Messages

Mag 847, Level 16, Share 0, Line 1

The statement has been terminated.

SQLQuery3.sql -JO...Voe Shepherd (56))* ** SQLQuery2.sql -JO...Voe Shepherd (55))

SQLQuery3.sql -JO...Voe Shepherd (56))* ** SQLQuery2.sql -JO...Voe Shepherd (55))

SQLQuery3.sql -JO...Voe Shepherd (56))* ** SQLQuery2.sql -JO...Voe Shepherd (55))

SQLQuery3.sql -JO...Voe Shepherd (56))* ** SQLQuery2.sql -JO...Voe Shepherd (55))

SQLQuery3.sql -JO...Voe Shepherd (56))* ** SQLQuery2.sql -JO...Voe Shepherd (55))

SQLQuery3.sql -JO...Voe Shepherd (56))* ** SQLQuery2.sql -JO...Voe Shepherd (55))

SQLQuery3.sql -JO...Voe Shepherd (56))* ** SQLQuery2.sql -JO...Voe Shepherd (55))

SQLQuery3.sql -JO...Voe Shepherd (56))* ** SQLQuery2.sql -JO...Voe Shepherd (55))

SQLQuery3.sql -JO...Voe Shepherd (56))* ** SQLQuery2.sql -JO...Voe Shepherd (55))

SQLQuery3.sql -JO...Voe Shepherd (56))* ** SQLQuery2.sql -JO...Voe Shepherd (55))

SQLQuery3.sql -JO...Voe Shepherd (56))* ** SQLQuery2.sql -JO...Voe Shepherd (55))

SQLQuery3.sql -JO...Voe Shepherd (56))* ** SQLQuery2.sql -JO...Voe Shepherd (55))

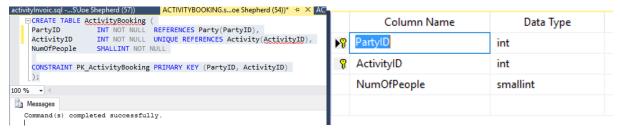
SQLQuery3.sql -JO...Voe Shepherd (56))* ** SQLQuery2.sql -JO...Voe Shepherd (55))

SQLQuery3.sql -JO...Voe Shepherd (56))* ** SQLQuery2.sql -JO...Voe Shepherd (55))

SQLQuery3.sql -JO...Voe Shepherd (56))* ** SQLQuery2.sql -JO...Voe Shepher
```

Activity Booking

Figure 10



This is the ActivityBooking table (figure 10), which has partyID and ActivityID as both primary and foreign keys that reference the party and activity table respectively. I add a NumOfPeople column to make it easier to work out how many people are participating in each activity. The activity ID is set to UNQIUE so that an activity cannot be booked more than once.

To insert into this table (figure 10.1) I simply put the partyID, ActivityID and a select statement that selects the partysize column from the party table as the number of people participating.

The only test data I inserted was to see if an activity can be booked more than once (figure 10.2). The screenshot shows that if someone is trying to book the same activity it cannot be done.

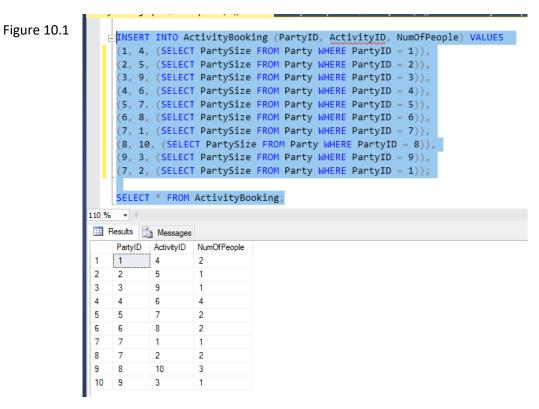


Figure 10.2

```
ENSERT INTO ActivityBooking (PartyID, ActivityID, NumOfPeople) VALUES

(1, 4, (SELECT PartySize FROM Party WHERE PartyID = 1)),

(2, 4, (SELECT PartySize FROM Party WHERE PartyID = 2));

SELECT * FROM ActivityBooking;

Results Messages

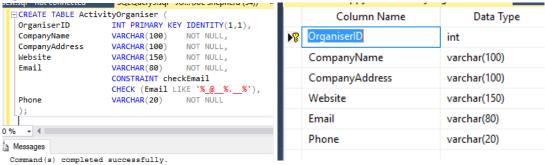
Msg 2627, Level 14, State 1, Line 11

Violation of PRIMARY KEY constraint 'PK_ActivityBooking'. Cannot insert duplicate key in object 'dbo.ActivityBooking'. The duplicate key value is The statement has been terminated.

(10 row(s) affected)
```

Activity Organiser

Figure 11



```
| ⊟ALTER TABLE ActivityOrganiser | ADD CONSTRAINT chkPhoneNumber CHECK (Phone NOT LIKE '%[a-z]%' AND DATALENGTH(Phone) < 13 AND DATALENGTH(Phone) > 10 );
```

This is the Activity Organiser table (figure 11). I added a check constraint named checkEmail which does the same thing as the chkEmail constraint on the guest table. I later added another constraint named chkPhoneNumber, which checks that no alphabetic characters are entered into this field, and that is the length of a phone number.

Figure 11.1



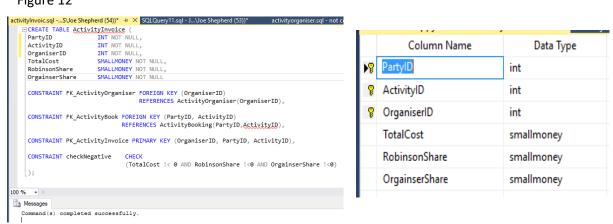
Figure 11.2

```
INSERT INTO ActivityOrganiser VALUES
('Lacys HorseRiding School' , '401 Aliquam Road','LacyHorses.com','Lacy_HorseHotmail.co.uk','01273 767524');
     SELECT * FROM ActivityOrganiser;
100 % - 4
Results Messages
   Msg 547, Level 16, State 0, Line 2
The INSERT statement conflicted with the CHECK constraint "checkEmail". The conflict occurred in database "Happy Vally Campsite", table "dbo.ActivityOrge
  The statement has been terminated.
      INSERT INTO ActivityOrganiser VALUES
('Lacys HorseRiding School', '401 Aliquam Road','LacyHorses.com','Lacy_Horse@Hotmail.co.uk','0127');
       SELECT * FROM ActivityOrganiser:
  Results 🛅 Messages
       g 547, Level 16, State 0, Line 2
e INSERT statement conflicted wi
                                             with the CHECK constraint "chkPhoneNumber". The conflict occurred in database "Happy Vally Campsite", table "dbo.Activity
     The statement has been terminated.
         INSERT INTO ActivityOrganiser VALUES
('Lacys HorseRiding School', '401 Aliquam Road','LacyHorses.com','Lacy_Horse@Hotmail.co.uk','01273 35445644424');
        SELECT * FROM ActivityOrganiser;
   Results 🛅 Messages
       Mag 547, Level 16, State 0, Line 2
The INSERT statement conflicted with the CHECK constraint "chlPhoneNumber". The conflict occurred in database "Happy Vally Campsite", table "dbo.Activity The statement has been terminated.
```

Figure 11.1 shows the insert statement to fill in the activity organiser table and the resulting table. Figure 11.2 shows the test data inserted to check the constraints on the table are working correctly. All the tests show an error with the constraint that is being enforced.

Activity Invoice

Figure 12



This is the Activity Invoice table (figure 12). There is once check constraint named chkNegative that checks to see if the data inserted into the totalCost, robinsonShare and OrganiserShare columns is not negative, as they cannot earn a minus amount of money from the activities.

Figure 12.1

```
tivityinice.sqi -...P5\Joe Snepnera (53))" 😕 🔨 activityorganiser.sqi - not connected
  INSERT INTO ActivityInvoice (PartyID,ActivityID,OrganiserID,TotalCost,RobinsonShare,OrgainserShare)
   SELECT PartyID , ActivityBooking.ActivityID, Activity.OrganiserID , 0, 0, 0
    FROM ActivityBooking, Activity, ActivityOrganiser
    WHERE Activity.ActivityID = ActivityBooking.ActivityID AND Activity.OrganiserID = ActivityOrganiser.OrganiserID;
   UPDATE ActivityInvoice
   SET TotalCost =
   PricePerPerson * NumOfPeople
   FROM Activity, ActivityBooking
   WHERE ActivityInvoice.ActivityID = Activity.ActivityID And Activity.ActivityID = ActivityBooking.ActivityID;
   UPDATE ActivityInvoice
   SET RobinsonShare =
   TotalCost*0.1:
   UPDATE ActivityInvoice
   SET OrgainserShare =
   TotalCost*0.9;
   SELECT * FROM ActivityInvoice ORDER BY PartyID;
```

To insert data into this table, I used a select statement to get the IDs from the activity booking, activity and activity organiser table that match the values for each row in the activity booking table. I then used the update statement that works out the total cost for each activity by multiplying the PricePerPerson and NumOfPeople column from the activity and activity booking table, and two more update statements to work out the robinson share and organiser share. This made it much easier to

work out the values in an automatic way rather than doing each row manually. The results of the query are shown below.

	PartyID	ActivityID	OrganiserID	TotalCost	RobinsonShare	OrgainserShare
1	7	1	1	20.00	2.00	18.00
2	1	4	2	40.00	4.00	36.00
3	4	6	2	120.00	12.00	108.00
4	5	7	3	20.00	2.00	18.00
5	9	3	3	10.00	1.00	9.00
6	2	5	4	15.00	1.50	13.50
7	6	8	6	10.00	1.00	9.00
8	3	9	8	40.00	4.00	36.00
9	7	2	9	50.00	5.00	45.00
10	8	10	10	60.00	6.00	54.00

Figure 12.2



Figure 12.2 shows the test data I inserted to see if I could have entered negative values in one of the columns, which returned an error.

Banned Guest

Figure 13



This is the final table banned guest. I added one check constraint to check if the person being banned cannot be banned from a date set in the past. Figure 13.1 shows the insert statement. I used a subquery to select the GuestID from the guest table with a where condition of firstname and lastname, as I figured if you are banning someone you would not know their guestID, only their name. The last screenshot shows the test data entered to show that the check constraint does not allow a guest to be banned from a date set in the past. An error was returned.

Figure 13.1

```
IED GUEST.sql...Joe Shepherd (53))* → SQLQuery3.sql - JO...VJoe Shepherd (56))* SQLQuery2.sql - JO...VJoe Shepherd (55))

SQLQuery1.sql - JO...VJoe Shepherd (55))

SQLQuery2.sql - JO...VJoe Shepherd (55))

SQLQuery1.sql - JO...VJoe Shepherd (55))

SQLQuery1.sql - JO...VJoe Shepherd (55))

SQLQuery2.sql - JO...VJoe Shepherd (55))

SQLQuery1.sql - JO...VJoe Shepherd (55))

SQLQuery1.sql - JO...VJoe Shepherd (55))

SQLQuery2.sql - JO...VJoe Shepherd (55))

SQLQuery1.sql - JO...VJoe Shepherd (55))

SQLQuery2.sql - JO...VJoe Shepherd (55)

SQLQuery2.sql - JO...VJoe Shepherd (55)

SQLQuery2.sql - JO...VJoe Shepherd (55)

SQLQuery1.sql - JO...VJoe Shepherd (55)

SQLCuery1.sql - JO...VJoe Shepherd (55)

SQLCuery1.sql - JO...VJoe Shepherd (55)

SQLECT GuestID FROM Guest WHERE FirstName = 'Dahla' AND LastName = 'Navaron'), '20170511', 'Littering'), '(SELECT GuestID FROM Guest WHERE FirstName = 'Willa' AND LastName = 'Walker'), '20170428', 'Littering'), '
```

	GuestID	DateBanned	Reasons
1	110	2017-05-11	Fighting with other guests
2	111	2017-05-11	Vandalsim
3	114	2017-05-05	Unpaid fee
4	119	2017-05-11	Littering
5	120	2017-05-11	Abuse towards staff
6	123	2017-05-20	Vandalism
7	126	2017-04-28	Littering
8	127	2017-05-15	Unpaid fee
9	128	2017-04-15	Abuse towards staff
10	129	2017-04-15	Fighting with other guests

```
BANNED GUEST.sql...Joe Shepherd (53))* *** SQLQuery3.sql - JO...Voe Shepherd (56))* SQLQuery2.sql - JO...Voe Shepherd (55)) SQLQuery1.sql - JO...Voe Shepherd (54))

1 SQLQuery2.sql - JO...Voe Shepherd (55))

2 VALUES

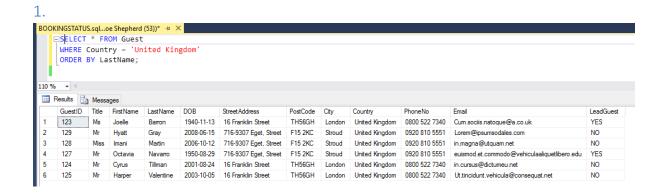
3 ((SELECT GuestID FROM Guest WHERE FirstName = 'Dahlia' AND LastName = 'Oliver'), '20170211', 'Fighting with other guests');

5

89 % 
4

1 The INSERT statement conflicted with the CMECK constraint "chiDateBanned". The conflict occurred in database "Happy Vally Campsite", table "dbo.BannedGuest", column 'DateBanned''. The statement has been terminated.
```

Scenarios and Queries



This is the first query from appendix B. It is a simple select statement that shows all the guests that live in the UK. To achieve this I used a WHERE clause.

2.

```
וואכ ואו כטוואכ Ari us.sqi...oe snepnera (ככן
 1 ☐ BEGIN TRANSACTION CreateBooking
     BEGIN TRY
  A DECLARE @guestid INT;
     DECLARE @partvid INT;
    INSERT INTO Guest VALUES
     ['Mr','Luke','Ribero','19940816', '18 Freshwater Avenue', 'RM6 4DT','London','United Kingdom', '07447400448', 'lukerib94@gmail.com', 'YES');
SELECT @guestId = SCOPE_IDENTITY();
 11
     INSERT INTO Party VALUES
       (4, 2, 2, 0, @guestId);
     SELECT @partyid = SCOPE IDENTITY();
      INSERT INTO PartyMembers VALUES (@guestId, @partyid);
     ☐ INSERT INTO Guest (Title, FirstName, LastName, DOB, StreetAddress, PostCode, City, Country, PhoneNo, Email) VALUES
      ('Mrs','Bella','Ribero','19960505', '18 FreshWater Avenue', 'RM6 4DT','London', 'United Kingdom', '07676345123', 'bellarib@gmail.com');

SELECT @guestid = SCOPE_IDENTITY();
      INSERT INTO PartyMembers VALUES (@guestid,@partyid);
      ('Mr','Paul','Ribero','20070528', '18 Freshwater Avenue', 'RM6 4DT','London', 'United Kingdom', '07676345123', 'paulisthebest@hotmail.com');
SELECT @guestid = SCOPE_IDENTITY();
 22
    insert into Guest (Title, FirstName, LastName, DOB, StreetAddress, PostCode, City, Country, PhoneNo, Email) VALUES
      INSERT INTO PartyMembers VALUES (@guestid,@partyid);
     INSERT INTO Guest (Title, FirstName, LastName, DOB, StreetAddress, PostCode, City, Country, PhoneNo, Email) VALUES
      ('Miss','Tara','Ribero','2011220', '18 FreshNater Avenue', 'RM6 4DT','London', 'United Kingdom', '07676345123', 'Tara123@gmail.com');

SELECT @guestid = SCOPE_IDENTITY();
      INSERT INTO PartyMembers VALUES (@guestid,@partyid);
      (@partyid, | (SELECT TOP 1 AccommodationID FROM Accommodation WHERE AccType = 'Motorhome' AND Capacity = 4),
 33
       '20170418', '20170423');
      COMMIT TRANSACTION CreateBooking:
      BEGIN CATCH
ROLLBACK TRANSACTION CreateBooking;
 41
      END CATCH;
```

For this query, a guest had to be added to the system and create a booking, recording the details of the party members. To achieve this I decided to use a transaction because if any of the entries had for some reason failed, none of the tables would be updated, which protects the data integrity otherwise there could have been incorrect data for the party members or booking table. By declaring two variables @GuestID and @PartyID, it made it very easy to insert the correct guestID

and PartyID into the party members table. After each guest had been added to the guest table, I used the SCOPE_IDENTITY function to select the last entered guestID and set @GuestID as this value. The party was created after the lead guest was added, and again using SCOPE_IDENTITY to set @PartyID as the last entered PartyID in the table. Then to add a guest to the party members table I only had to use the variables, instead of searching for their ID from the table and adding it afterwards. To create the booking, I again used the @PartyID variable and a select subquery to select the accommodation that the guests required with the capacity of their party size. The results of this query are show below highlighted in blue to show the guests and booking that was created.

		-									
GuestID	Title	FirstName	LastName	DOB	Street Address	PostCode	City	Country	PhoneNo	Email	LeadGuest
137	Mr	Luke	Ribero	1994-08-16	18 FreshWater Avenue	RM6 4DT	London	United	07447400448	lukerib94@gmail.com	YES
138	Mrs	Bella	Ribero	1996-05-05	18 FreshWater Avenue	RM6 4DT	London	United	07676345123	bellarib@gmail.com	NO
139	Mr	Paul	Ribero	2007-05-28	18 FreshWater Avenue	RM6 4DT	London	United	07676345123	paulisthebest@hotmail.com	NO
140	Miss	Tara	Ribero	2010-12-20	18 FreshWater Avenue	RM6 4DT	London	United	07676345123	Tara123@gmail.com	NO

PartyID	PartySize	Adults	Children	Pets	LeadGuestID
10	3	1	2	0	127
12	4	2	2	0	137

	GuestID	PartyID
21	137	12
22	138	12
23	139	12
24	140	12

BookingID	PartyID	AccommodationID	CheckIn	CheckOut	NumOfNights
25	10	27	2017-04-10	2017-04-15	5
28	12	27	2017-04-18	2017-04-23	5

```
Query6.sql - JO...\Joe Shepherd (56))* 3.sql - JOESHEPS.H...\Joe Shepherd (54))* → ×
      BEGIN TRY
       DECLARE @guestid INT:
       DECLARE @partyid INT;
       ('Mr','David','Jones','19820306', '39 Brookside', 'EN4 8TT','London','United Kingdom', '0782678392' , 'DJones@trentschool.org', 'YES');

SELECT @guestId = SCOPE_IDENTITY();
       INSERT INTO Party VALUES
       (10, 3, 7, 0, @guestId);
SELECT @partyid = SCOPE_IDENTITY();
 12
 13
       INSERT INTO PartyMembers VALUES (@guestId, @partyid);
 17
       INSERT INTO Guest (Title, FirstName, LastName, DOB, StreetAddress, PostCode, City, Country, PhoneNo, Email) VALUES
       ('Miss','Amelia','Scott','19700509', '1 littlegrove', 'EN4 89T','London', 'United Kingdom', '0776354789', 'Ascott@trentschool.org');

SELECT @guestid = SCOPE_IDENTITY();
 18
 19
       INSERT INTO PartyMembers VALUES (@guestid,@partyid);
 21
       INSERT INTO Guest (Title, FirstName, LastName, DOB, StreetAddress, PostCode, City, Country, PhoneNo, Email) VALUES

('Mr','Fred','Henderson','19730816', '18 Churchhill Road', 'ENS 7NR','London', 'United Kingdom', '07662333456' , 'Fhenderson@trentschool.org');

SELECT @guestid = SCOPE_IDENTITY();
 24
 25
       INSERT INTO PartyMembers VALUES (@guestid,@partyid);
 26
Query6.sql - JO...\Joe Shepherd (56))* 3.sql - JOESHEPS.H...\Joe Shepherd (54))* → ×
      INSERT INTO Guest (Title, FirstName, LastName, DOB, StreetAddress, PostCode, City, Country, PhoneNo, Email) VALUES
       ('Miss','Kim','Kardasian','20011012', '52 Bournechurch Road', 'EN4 SYT','London', 'United Kingdom', '02084470036', 'Kimmygirl@gmail.com');
SELECT @guestid = SCOPE_IDENTITY();
 32
       INSERT INTO PartyMembers VALUES (@guestid,@partyid);
      INSERT INTO Guest (Title, FirstName, LastName, DOB, StreetAddress, PostCode, City, Country, PhoneNo, Email) VALUES

('Miss','Michelle','Simpson','20010318', '57 Bournechurch Road', 'EN4 SYT','London', 'United Kingdom', '02088876354', 'Michsimpson@gmail.com');

SELECT @guestid = SCOPE_IDENTITY();
 37
       INSERT INTO PartyMembers VALUES (@guestid,@partyid);
      INSERT INTO Guest (Title, FirstName, LastName, DOB, StreetAddress, PostCode, City, Country, PhoneNo, Email) VALUES

('Mr','Jake','Simpson','20010318', '57 Bournechurch Road', 'EN4 SYT','London', 'United Kingdom', '02088876354' , 'Jakesimpson@gmail.com');

SELECT @guestid = SCOPE_IDENTITY();
 39
 40
 41
       INSERT INTO PartyMembers VALUES (@guestid,@partyid);
 42
      ('Miss','Frankie','Holiday','20010504', '25 Milner Road', 'ENG 7HY','London', 'United Kingdom', '02089987655', 'Frankie001@gmail.com');
SELECT @guestid = SCOPE_IDENTITY();
 47
       INSERT INTO PartyMembers VALUES (@guestid,@partyid);
 48
      INSERT INTO Guest (Title, FirstName, LastName, DOB, StreetAddress, PostCode, City, Country, PhoneNo, Email) VALUES

('Mr','Liam','Moony','20010917', '11 Avendale Av', 'EN3 3XR','London', 'United Kingdom', '02084432256', 'FootballKing@gmail.com');

SELECT @guestid = SCOPE_IDENTITY();
 51
       INSERT INTO PartyMembers VALUES (@guestid,@partyid);
      EINSERT INTO Guest (Title, FirstName, LastName, DOB, StreetAddress, PostCode, City, Country, PhoneNo, Email) VALUES
['Mr','Ryan','Crossley','20011129', '82 Cresent Road', 'ENS 9JJ','London', 'United Kingdom', '02089967836', 'Fatboyryan@gmail.com');

SELECT @guestid = SCOPE_IDENTITY();
 57
       INSERT INTO PartyMembers VALUES (@guestid,@partyid);
 58
                           INSERT INTO Booking VALUES
                            (@partyid,

(SELECT TOP 1 AccomodationID FROM Accomodation WHERE AccType = 'Tent' AND Capacity = 6),

'20170601', '20170608');
                             INSERT INTO Booking VALUES
                            (@partyid, (@partyid, Capacity = 1), Capacity = 4), '20170601', '20170601', '20170608');
                     73
                            COMMIT TRANSACTION CreateBooking2;
                            END TRY
BEGIN CATCH
ROLLBACK TRANSACTION CreateBooking2;
END CATCH;
                     81
                          - 4 1
                    row(s) affected)
                   l row(s) affected)
```



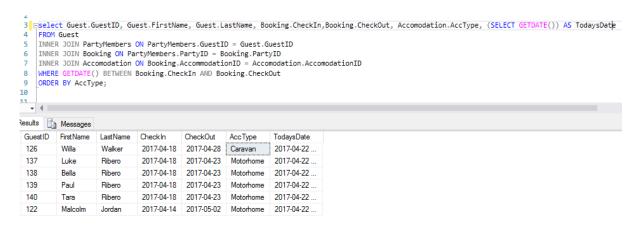
PartyID	PartySize	Adults	Children	Pets	LeadGuestID
9	1	1	0	2	126
10	3	1	2	0	127
12	4	2	2	0	137
16	10	3	7	0	173

BookingID	PartyID	AccommodationID	CheckIn	CheckOut	NumOfNights
28	12	27	2017-04-18	2017-04-23	5
104	16	29	2017-06-01	2017-06-08	7
105	16	26	2017-06-01	2017-06-08	7

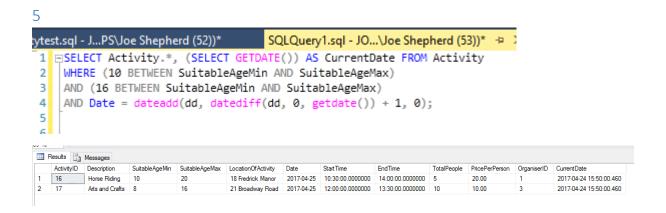
GuestID	PartyID
140	12
173	16
174	16
175	16
176	16
177	16
178	16
179	16
180	16
181	16
182	16

This query is essentially the same as query 2, however more guests are added as it is for a school party. I could not manage to allow multiple pitches to be assigned to one booking, so I had to create two bookings with the same partyID. I realised to do this query I could have maybe added another table named BookedAccommodation, which references the BookingID from the booking table. Then removed the accommodationID from the booking table, moving it to the BookedAccomodation table to show that a booking can have more than one accomodationID in the booking. So, the new table would have been used as an intersection table. However, because I had created the whole database and entered all the data into it, this would have been too time consuming.

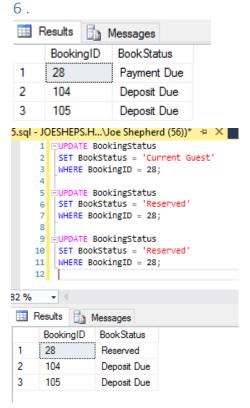
4.



Query 4 was to list the guests staying overnight tonight, grouped by accommodation type. To do this I selected the guest's details from the guest table, the checkin and checkout dates from the booking table, and the acctype from the accommodation table. I then used three inner joins to the guest table. The first being the party members table to match the guest ids together, so we know what guest is associated with what party. The second table being the booking table to match the party ids together, so know we know what party is associated with what booking. Thirdly, the accommodation table to match the accommodation ids together, so know we know what accommodation type has been booked out. Then by using a WHERE clause to only return the guests who are booked in over the current date returns which guests are staying overnight tonight.



To return activities suitable for a 10-16 year old, a simple select statement with a WHERE clause to check if 10 or 16 is between the minAge and maxAge, and the dateadd function to search for activities that are on tomorrow is able return the data required. The dateadd function truncates the time from the getdate function because just using date = GETDATE does not work. The getdate + 1 gives tomorrows date. I added a current date column at the end to prove only activities set for tomorrow are returned.



1 □ SELECT

FROM Invoice

Messages

MoneyPaid

2005.00

MoneyDue

2175.00

3

5

8

10

13

15

Results

20

TotalGuests

81 %

For this query, the booking status had to be updated. To do this an UPDATE statement is all that is needed, setting the bookStatus to the value specified, using a WHERE clause to select which guests booking status needs to be updated.

The picture at the top shows the booking status before it was updated.

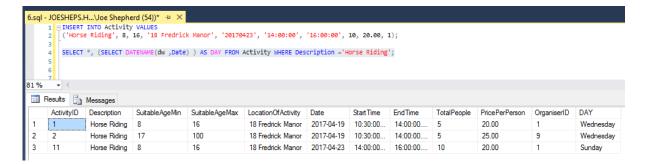
```
6.sql - JOESHEPS.H...\Joe Shepherd (54))* → ×
         (SELECT COUNT(GuestID) FROM PartyMembers
         INNER JOIN Booking ON Booking.PartyID = PartyMembers.PartyID
             (Booking, CheckIn BETWEEN '20170401' AND '20170430')
     6 OR (Booking.CheckOut BETWEEN '20170401' AND '20170430')) AS TotalGuests,
        SUM(Invoice.TotalPaid) AS MoneyPaid,
     9 SUM ( Invoice.TotalPrice - Invoice.TotalPaid) AS MoneyDue
         INNER JOIN Booking ON Booking.BookingID = Invoice.BookingID
    12 WHERE (Booking.CheckIn BETWEEN '20170401' AND '20170430')
         OR (Booking.CheckOut BETWEEN '20170401' AND '20170430');
```

This query was to find out the total guests staying in the last month, and to find out the total money paid and total money due. To achieve this, I used a subquery for the TotalGuests column. This subquery uses the count function to count the number of people from the party members table, who have stayed at the campsite in the last month. To find out what guests stayed in the last month I had to inner join the booking table, matching the partyIDs and used a WHERE clause that returns the people who have a checkin or checkout date that is between the month of April.

7.

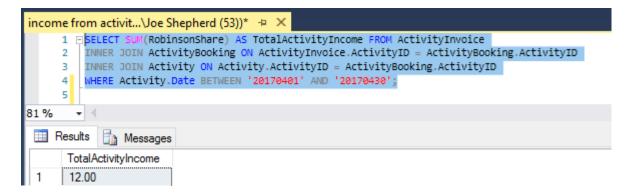
To get the total money paid, I used the SUM function to add the TotalPaid column from invoice, and a sum of the TotalPrice minus the TotalPaid column in the invoice table for the money due column. Again, using an inner join on the booking table like before to add up only the guests who have stayed in the last month.

8.



This query had to add a new horse riding activity to the rota between 2-4pm on a sunday. This was just a simple insert statement. I added DAY column at the end of the table to prove the activity was added on the following Sunday.

9.

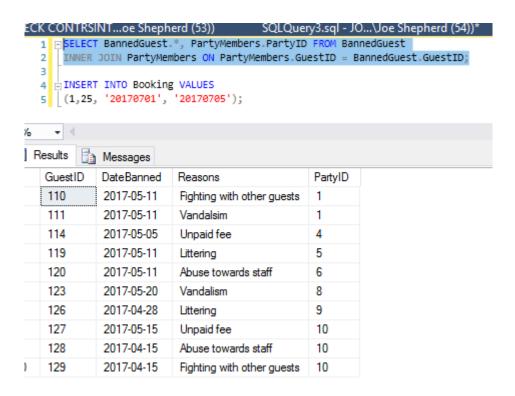


This query had to return the total income from activities in the month of January. I did not have any activities that happened in the month of January, so I used April instead. I used the SUM function to add up the RobinsonShare column in the activity invoice table, and used an inner join to join the activity booking table by activityID, so that I could use a WHERE clause on the activity booking table to return only the activities that were booked in April.

```
10.
                                                     SQLQuery2.sql - JO...\Joe Shepherd (53))* 😕 🔾
        ncome from activit...\Joe Shepherd (54))
             1 ☐ SELECT
                     Guest.FirstName + ' ' + Guest.LastName AS Name, Booking.NumOfNights ,
             2
                     Party.PartySize - 1 AS AdditonalGuests
             3
                 FROM
             5
                     Booking
                 INNER JOIN Party ON Party.PartyID = Booking.PartyID
             6
                 INNER JOIN Guest ON Guest.GuestID = Party.LeadGuestID
             8
                 WHERE Guest.FirstName = 'Quemby' AND Guest.LastName='Fitzpatrick' ;
             9
               + 4
                    Messages Messages
        Results
                                 NumOfNights
                                               AdditonalGuests
                                               3
              Quemby Fitzpatrick
```

This query had to find the total number of nights, and additional party members, for a given guest. This was achieved by joining two tables to the booking table. The first join being the party table by matching PartyID to the Booking table, then joining the guest table by GuestID on LeadGuestID. This finds out what party the lead guest is associated with, and uses the partysize column -1 to find additional guests, as the guest in question is not an additional guest so 1 had to be taken away from the party size.

11.



This query was to find out what guests have cause trouble in the past, and ensure they cannot make a booking again. To return the guests that cause trouble this was just a simple select statement, joining the Party table to show what party they are associated with. However, to make sure that they could never book again, I created a trigger on the booking table as seen below.

```
HECK CONTRSINT...oe Shepherd (53))
                                        BANNED GUEST TRIG...oe Shepherd (54))
       CREATE TRIGGER Banned ON dbo.Booking
       after INSERT, UPDATE
       AS
   3
   5
      8
           SELECT 1 FROM inserted as x
        INNER JOIN PartyMembers ON x.PartyID = PartyMembers.PartyID
  10
       INNER JOIN BannedGuest ON PartyMembers.GuestID = BannedGuest.GuestID
  11
  12
  13
  14
       BEGIN
  15
        RAISERROR('Banned Guest is trying to make a booking', 16, 1)
  16
  17
        ROLLBACK TRANSACTION;
  18
        RETURN
  19
        END
  20
  21
Messages
 Command(s) completed successfully.
```

This trigger is for after data is inserted or updated in the booking table. It checks to see if a guest who is banned is part of a party that is trying to make a booking, it will come up as an error and not

allow the booking to go through. The result of executing the insert statement in the first screen shot produces this error below.

```
Messages

Msg 50000, Level 16, State 1, Procedure Banned, Line 16 [Batch Start Line 3]
Banned Guest is trying to make a booking
Msg 3609, Level 16, State 1, Line 4
The transaction ended in the trigger. The batch has been aborted.
```

Triggers/ Stored Procedures and Functions

Α.

The first stored procedure is required that will create a new booking using parameters giving the detail of the date and time, the main guest and party members, and the type of accommodation required. To do this I first created another stored procedure to automatically work out the values that needed to be inserted into the invoice table.

This procedure is the same as the insert statement I used to insert the data into the Invoice table. It works out the total price of the booking, the deposit and sets the total paid as either the deposit price for advance bookings or the full price for on the day bookings.

```
CREATE PROCEDURE WorkoutPayment
22
    @BookingID INT
    BEGIN
24
25
   INSERT INTO Invoice VALUES
26
27
        (SELECT SUM(NumOfNights * PricePerNight) FROM Booking, Accommodation WHERE Booking.BookingID = @BookingID AND
28
29
         Accomodation.AccomodationID = Booking.AccommodationID),
31
32
33 FIF (SELECT Booking CheckIn FROM Booking WHERE BookingID = @BookingID) > GETDATE()
34 DUPDATE Invoice
       SET Deposit = TotalPrice * 0.25,
35
36
        TotalPaid = TotalPrice * 0.25
37
        WHERE BookingID = @BookingID
39
40
```

```
Query4.sql - JO...\Joe Shepherd (53))* 垣 🔀
 1 □ CREATE PROCEDURE CreateBooking
    @partyid INT,
    @checkin DATE,
    @checkout DATE,
    @acctype VARCHAR(30)
 5
 6
 7
     BEGIN
 8
 9 | INSERT INTO Booking VALUES
        (@partyid,
10
       (SELECT TOP 1 Accomodation.AccomodationID FROM Accomodation
11
         INNER JOIN Party ON Party.PartySize <= Accomodation.Capacity
12
13
         WHERE Party.PartyID = @partyid AND Accomodation.AccType = @acctype
         ORDER BY Accomodation.Capacity),
14
15
         @checkin,
16
         @checkout);
17
     DECLARE @BookingID INT = SCOPE_IDENTITY();
18
     EXEC WorkoutPayment @BookingID;
19
20
21 INSERT INTO BookingStatus VALUES
         (@BookingID, 'Reserved');
22
 23
     END;
24
25
```

Above is the Stored Procedure that creates a booking. It takes in parameters of the PartyID (which is essential the main guest and party members information), the check in and check out date, and the accommodation type. To find the accommodationID for the booking, I used a sub query that selects the top 1 accommodation, where the accommodation type is the same as the input parameter @acctype, and the capacity is greater than or equal to the party size. To achieve this an inner join was necessary to match the accommodation capacity to the party size.

Then I called the workout payment stored procedure to insert the data into the invoice table automatically from the booking id that was just created. Executing the query produces the results shown below.

EXECUTE CreateBooking 12, '20170609', '20170615', 'Tent'; SELECT * FROM Booking; SELECT * FROM Invoice;										
	CT * FROM Bo		us;							
		nessages								
	BookingID	PartyID	/ID AccommodationID		Checkl	n	Check	cOut	NumOfNights	
13	105	16	26		2017-0	2017-06-01		-06-08	7	
14	115	12	26		2017-0	6-09 2017		-06-15	6	
	AccInvoicel		BookingID TotalPri		Deposit		TotalPaid			
13	163	105		105.00	26.25	26.2	25			
14	166	115		90.00	22.50	22.5	50			
	BookingID	BookSta	tus							
13	105	Deposit	Due							
14	115	Reserve	d							

```
LQuery4.sql - JO...\Joe Shepherd (53))*   □  ×
    1 □CREATE PROCEDURE ActivityList
      AS
    3 ⊨BEGIN
    4
    5
            SELECT
    6
    7
            Activity.ActivityID,
            Activity.Description,
    8
    9
            TotalPeople - NumOfPeople AS Spaces
   10
            FROM Activity
   11
            INNER JOIN ActivityBooking ON Activity.ActivityID = ActivityBooking.ActivityID
            WHERE Activity.Date >= GETDATE()
   12
            ORDER BY Activity.ActivityID;
   13
   14
   15
            SELECT Activity.ActivityID , Guest.FirstName +' '+ Guest.LastName AS Participant
   16 📋
            FROM Activity
   17
   18
            INNER JOIN Activitybooking oN ActivityBooking.ActivityID = Activity.ActivityID
            INNER JOIN PartyMembers ON ActivityBooking.PartyID = PartyMembers.PartyID
   19
            inner join guest ON Guest.GuestID = PartyMembers.GuestID
   20
            WHERE Activity.Date >= GETDATE()
   21
            order by ActivityID;
   22
   23
   24
        END;
   25
```



Hamer Val...

12 10

This procedure was to produce an activity list with the participants involved, indicating any spaces on the activities. To do this I had to create two separate tables, one showing the number of people, and the spaces in each activity, and another table showing who is involved in each activity.

To return how many spaces are left in each activity, I used the totalpeople – numofpeople from the activity and activity booking table respectively using a join on the activityID to match the tables together.

To get each participate in each activity I had to use three joins on the activity table. The first was the activity booking table, which matches the activity to the booking, then another join on the party members, to match which party member is in each party, and finally the guest table, to match the guest's details to each party. From this It then returns the names of the guests in each activity. The results are shown to the left.

C.

```
ityinice.sql -...PS\Joe Shepherd (58))
                                       JOESHEPS.Happy Val...site - dbo.Booking
                                                                                   SQLQuery6
  1 □ CREATE TRIGGER CanceledBooking ON Booking
  2
      FOR DELETE
 3
     AS
  4 BEGIN
  5
  6
  7
          DELETE FROM ActivityInvoice
  8
          WHERE ActivityInvoice.PartyID = (SELECT deleted.PartyID FROM deleted)
  9
 10
          DELETE FROM ActivityBooking
          WHERE ActivityBooking.PartyID = (SELECT deleted.PartyID FROM deleted)
 11
 12
 13
      END ;
 14
 15
```

This is a trigger which is actioned when a booking is cancelled. This will check if the party have made any additional activity bookings and remove these from the system. To achieve this the Trigger is set to FOR DELETE, and then deletes any data from the activity invoice and activity booking table, that have a matching party ID as the party ID in the booking that is being cancelled. The activity invoice table had to be delete first as this references the activity booking table. Results of before and after are shown below. You can see that the party ID being deleted from the booking is no longer in the activity booking or activity invoice table.

