Ahmed El-Hawat
21831369

University Of Brighton

Databases Management Systems

Bet's Pets' Vets

Part B – Implementation

GitHub Repo - https://github.com/aeh73/DBMS

Table of Contents

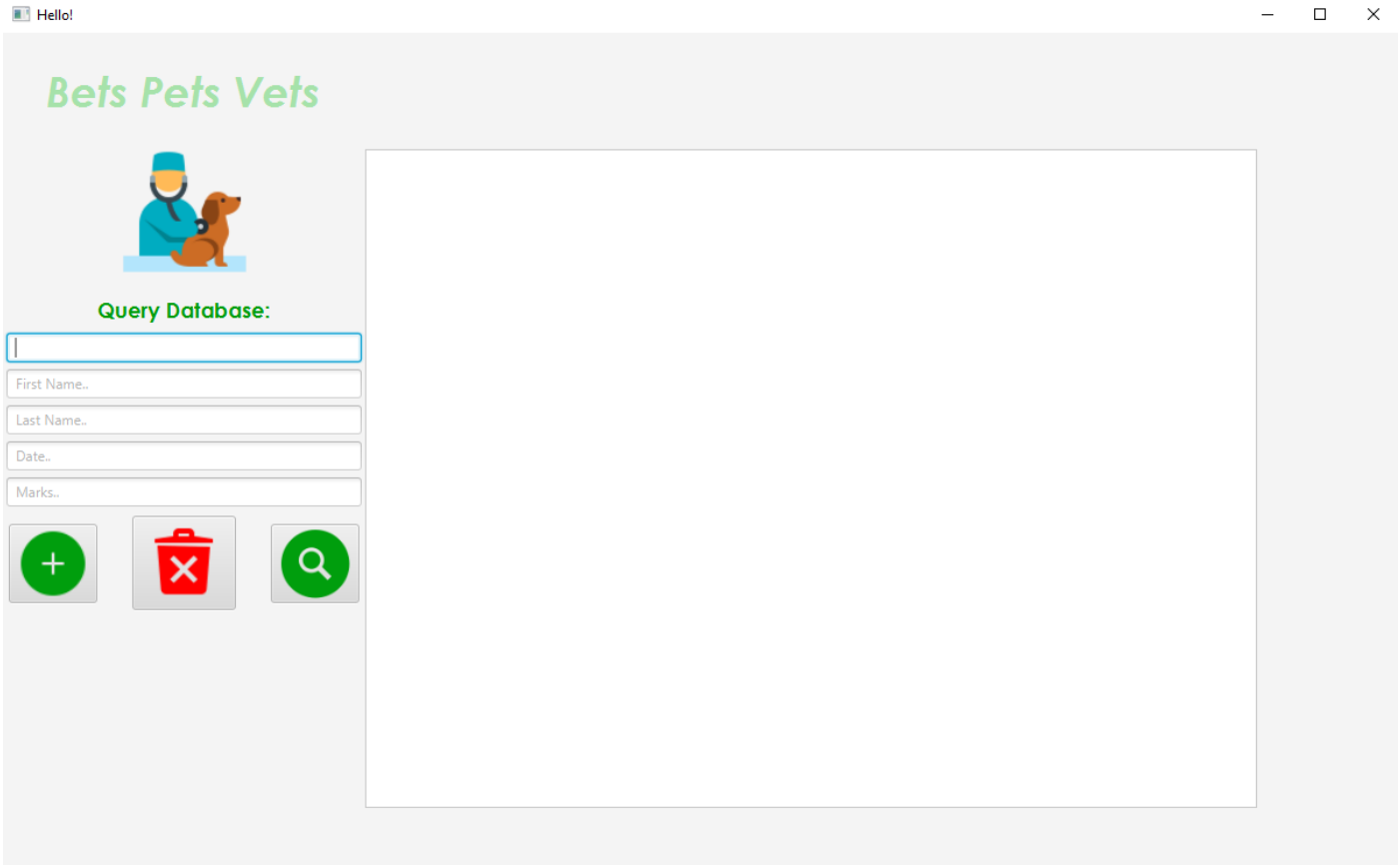
Introduction

In this project I will attempt to construct a practical database for “Bets Pets Vets” that meet the system requirements prescribed, to do this I will be using the entity-relationship diagram (ERD) provided with some minor adjustments and I will be using SQL Server Management Studio (SSMS) as well as LocalDB which comes with the Express version of SSMS.

I’ve chosen to use LocalDB for a few reasons.

i. Ease of setup and lightweight – Easy and quick installation, much smaller than the full version of SSMS, requires less resources than the full version as the database is launched on demand rather than run as a service.
ii. Fully capable – Has the exact same abilities and functionality of the full version of SQL server, while not intended for large scale databases it is great for small scale development and testing.
iii. Fully compatible – Fully compatible with other SQL Server editions supporting the same T-SQL language and programming model which can provide a smooth transition from development to production.

Finally, I’d like to create a simple user-friendly graphical interface using JavaFX and SceneBuilder and then connect the database to it using Java Database Connectivity (JDBC) drivers which will be able to run the prescribed queries.



Create Tables using SQL DDL

```
-- Create tblStaffType
CREATE TABLE tblStaffType (
    staffTypeID INT IDENTITY(1,1) PRIMARY KEY,
    staffTypeDesc VARCHAR(10) NOT NULL
);

-- Create tblPaymentType
CREATE TABLE tblPaymentType (
    paymentTypeID INT IDENTITY(1,1) PRIMARY KEY,
    paymentTypeDesc VARCHAR(20) NOT NULL
);

-- Create tblPetTypes
CREATE TABLE tblPetTypes (
    petTypeID CHAR(2) PRIMARY KEY,
    petTypeDesc VARCHAR(20) NOT NULL
);

-- Create tblOwners
CREATE TABLE tblOwners (
    ownerID INT IDENTITY(1,1) PRIMARY KEY,
    ownerFName VARCHAR(25) NOT NULL,
    ownerLName VARCHAR(25) NOT NULL,
    address VARCHAR(500)
);

-- Create tblPets
CREATE TABLE tblPets (
    petID INT IDENTITY(1,1) PRIMARY KEY,
    petName VARCHAR(25) NOT NULL,
    petTypeID CHAR(2) FOREIGN KEY REFERENCES tblPetTypes(petTypeID),
    ownerID INT FOREIGN KEY REFERENCES tblOwners(ownerID),
    petDOB DATE,
    petComments VARCHAR(1000)
);
```

Figure 1 CREATE Statement

CREATE TABLE tblStaffType creates the table 'tblStaffType' with 2 columns – **staffTypeID** and **staffTypeDesc**.

The **staffTypeID** is the primary key for the table and is an auto-incrementing integer with the 'IDENTITY(1,1)' meaning to start from 1 and increment by 1 each time a new staff type is added.

staffTypeDesc is a VARCHAR string with a maximum length of 10 and is 'NOT NULL' meaning if a new staff type is added it will need a description.

tblPaymentType, *tblPetTypes* and *tblOwners* follow the same pattern with differences in maximum length of descriptions and **petTypeID** being a string cannot be auto-incrementing in the same way as an integer ID like **staffTypeID** or **paymentTypeID** so will need to be hard-coded in for new pet types.

CREATE TABLE tblPets also has an auto-incrementing integer starting from 1 for its primary key, an VARCHAR with a maximum length of 25 for pet names, a CHAR with a maximum length of 2 to represent pet types, an INT to represent the owner ID and finally a DATE which will represent the pets date of birth and an VARCHAR for pet comments that has a maximum length of a thousand.

Both the **petTypeID** and **ownerID** are foreign keys referencing their respective tables primary keys.

```
-- Create tblStaff
CREATE TABLE tblStaff (
    staffID INT IDENTITY(1,1) PRIMARY KEY,
    staffFName VARCHAR(25) NOT NULL,
    staffLName VARCHAR(25) NOT NULL,
    staffTypeID INT FOREIGN KEY REFERENCES tblStaffType(staffTypeID),
    lineManagerID INT FOREIGN KEY REFERENCES tblStaff(staffID)
);

-- Create tblSurgeries
CREATE TABLE tblSurgeries (
    surgeryID INT IDENTITY(1,1) PRIMARY KEY,
    surgeryCode VARCHAR(2) NOT NULL,
    surgeryDetails VARCHAR(100) NOT NULL
);

-- Create tblTreatments
CREATE TABLE tblTreatments (
    treatmentCode CHAR(3) PRIMARY KEY,
    treatmentName VARCHAR(50) NOT NULL,
    treatmentDescription VARCHAR(100),
    treatmentCost DECIMAL(5,2) NOT NULL CHECK (treatmentCost >= 0),
    staffRequired VARCHAR(15) NOT NULL
);

-- Create tblAppointments
CREATE TABLE tblAppointments (
    appointmentID INT IDENTITY(1,1) PRIMARY KEY,
    staffID INT FOREIGN KEY REFERENCES tblStaff(staffID),
    petID INT FOREIGN KEY REFERENCES tblPets(petID),
    treatmentCode CHAR(3) FOREIGN KEY REFERENCES tblTreatments(treatmentCode),
    surgeryID INT NULL FOREIGN KEY REFERENCES tblSurgeries(surgeryID),
    date DATE NOT NULL,
    time TIME NOT NULL,
    outcome VARCHAR(100),
    invoiceNumber VARCHAR(10) UNIQUE NULL,
    paidStatus CHAR(1) CHECK (paidStatus IN ('Y', 'N')) NOT NULL DEFAULT 'N'
);

-- Create tblPayments
CREATE TABLE tblPayments (
    paymentID INT IDENTITY(1,1) PRIMARY KEY,
    ownerID INT FOREIGN KEY REFERENCES tblOwners(ownerID),
    invoiceNumber VARCHAR(10) FOREIGN KEY REFERENCES tblAppointments(invoiceNumber),
    paymentAmount DECIMAL(5,2) NOT NULL CHECK (paymentAmount >= 0),
    paymentTypeID INT FOREIGN KEY REFERENCES tblPaymentType(paymentTypeID),
```

Figure 2 CREATE Statement

CREATE tblStaff, *tblSurgeries* and *tblTreatments* again look very similar to the first set of tables with a couple exceptions, firstly we have a self-referencing foreign key in the *tblStaff* table, this makes **lineManagerID** dependant on the **staffID** itself which creates a hierarchical relationship within the staff table – this could get complicated depending how big your hierarchy is but as we only have 1 line manager it should be fine.

Next, we have a different data type in the *tblTreatments* with the **treatmentCost** which is 'DECIMAL(5,2)' this denotes the data will have a decimal point, 5 digits total and 2 digits after the decimal point (This gives a maximum treatment cost of 999.99), we also have a 'CHECK' constraint on the cost which makes the cost have to be equal or greater than zero – this potentially allows for a free treatment but cannot be negative.

CREATE tblAppointments and *tblPayments* also have a couple of differences, in *tblAppointments* we have **paidStatus** with a 'CHECK' constraint which forces the status to either be 'Y' or 'N', NOT NULL and by default be 'N' and in *tblPayments* we have a check constraint on the **paymentAmount** where the amount must be greater or equal to zero, the zero allowing for payment at a later date.

And this is a screenshot after the tables were created:

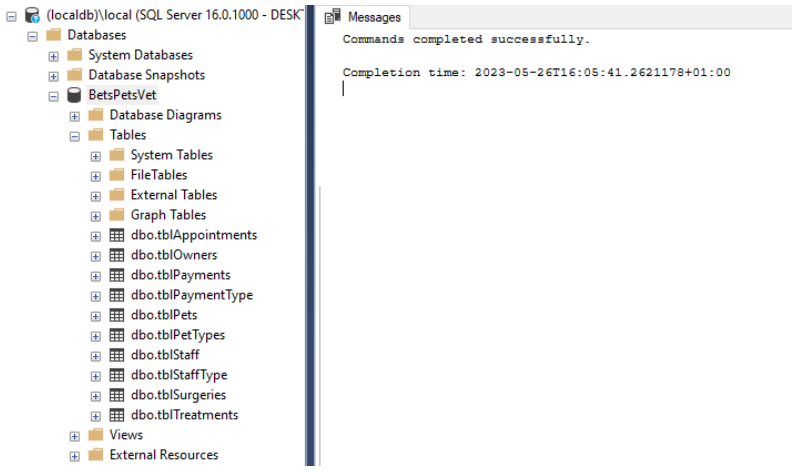


Figure 3 CREATE Execution

Populate Tables using SQL INSERT

This is my INSERT script separated into two screenshots, as I'm going through the system requirements, I'm sure I'll be inserting more data.

```
-- Insert into tblStaffType
INSERT INTO tblStaffType (staffTypeDesc) VALUES ('VET'), ('VET NURSE');

-- Insert into tblPaymentType
INSERT INTO tblPaymentType (paymentTypeDesc) VALUES
('Cash'),
('Credit Card'),
('Cheque');

-- Insert into tblPetTypes
INSERT INTO tblPetTypes (petTypeID, petTypeDesc) VALUES
('CT', 'cat'),
('DG', 'dog'),
('HR', 'horse'),
('RB', 'rabbit'),
('RD', 'rodent'),
('OT', 'other');

-- Insert into tblOwners
INSERT INTO tblOwners (ownerFName, ownerLName) VALUES
('Trevor', 'Tatler'),
('Bettina', 'Bishop'),
('Sally', 'Smith'),
('Yolande', 'Yearling'),
('Paul', 'Preston'),
('Praney', 'Patel');

-- Insert into tblPets
INSERT INTO tblPets (petName, petTypeID, ownerID, petDOB) VALUES
('Ratty', 'RD', 1, NULL),
('Carrots', 'RB', 2, NULL),
('Pinchbeck', 'HR', 3, NULL),
('Sleeky', 'CT', 4, NULL),
('Rover', 'DG', 5, NULL),
('Bugs', 'RB', 6, NULL);
```

Figure 4 INSERT Statement

```
-- Insert into tblStaff
INSERT INTO tblStaff (staffFName, staffLName, staffTypeID) VALUES
('Gurkiran', 'Greenwell', 1),
('Sarah', 'Smith', 2);

-- Insert into tblSurgeries
INSERT INTO tblSurgeries (surgeryCode, surgeryDetails) VALUES
('WG', 'Worthing Surgery Montague Place'),
('BN', 'Brighton Surgery Old Steine Place'),
('HV', 'Home Visit');

-- Insert into tblTreatments
INSERT INTO tblTreatments (treatmentCode, treatmentName, treatmentDescription, treatmentCost, staffRequired) VALUES
('T40', '30 min vet', 'Home Visit', 100, 'Vet'),
('T41', 'D-VAX-1', 'Dog - first dose standard' , 70, 'Vet or Nurse'),
('T42', 'D-VAX-2', 'Dog - booster annual standard dog', 50, 'Vet or Nurse'),
('T43', 'C-VAX-1', 'Cat - first dose standard', 60, 'Vet or Nurse'),
('T44', 'C-VAX-2', 'Cat - booster annual standard', 35, 'Vet or Nurse'),
('T45', 'diagnostic', 'Examination to determine cause and suggest treatment', 45, 'Vet'),
('T46', 'D-teeth', 'Dog - examine and monitor teeth / mouth', 30, 'Vet or Nurse'),
('T47', 'test', 'Biopsy, blood test etc.', 33, 'Vet or Nurse'),
('T48', 'Minor Operation', 'Small op under local anesthetic', 150, 'Vet');

-- Insert into tblAppointments
INSERT INTO tblAppointments (staffID, petID, treatmentCode, surgeryID, date, time, outcome, invoiceNumber, paidStatus) VALUES
(1, 1, 'T47', 1, '2022-11-13', '09:00', 'biopsy sent', 'FFF1234', 'Y'),
(1, 2, 'T45', 1, '2022-11-13', '11:45', 'Annual check - no issues', 'GGG1234', 'N'),
(1, 3, 'T40', 3, '2022-11-13', '13:15', NULL, 'HHH1234', 'N'),
(1, 4, 'T45', 2, '2022-11-13', '15:00', NULL, 'III1234', 'N'),
(1, 1, 'T48', 1, '2022-11-13', '19:00', NULL, 'JJJ1234', 'N'),
(2, 5, 'T46', 1, '2022-11-13', '10:45', 'Advised chew sticks', 'CCC1234', 'N'),
(2, 6, 'T47', 1, '2022-11-13', '13:30', NULL, 'DDD1234', 'N');

-- Insert into tblPayments
INSERT INTO tblPayments (ownerID, invoiceNumber, paymentAmount, paymentTypeID, paymentDate) VALUES
(1, 'FFF1234', 33, 2, '2022-11-13');
```

Figure 5 INSERT Statement

```
Messages

(2 rows affected)

(3 rows affected)

(6 rows affected)

(6 rows affected)

(6 rows affected)

(2 rows affected)

(3 rows affected)

(9 rows affected)

(7 rows affected)

(1 row affected)

Completion time: 2023-05-27T15:47:51.7310042+01:00
```

Figure 6 INSERT Execution

And the proof with the number of rows affected as well as a SELECT statement to show tblAppointments and its newly inserted data.

```
SELECT * FROM tblAppointments
```

100 %

Results | Messages

	appointmentID	staffID	petID	treatmentCode	surgeryID	date	time	outcome	invoiceNumber	paidStatus
1	1	1	1	T47	1	2022-11-13	09:00:00.0000000	biopsy sent	FFF1234	Y
2	2	1	2	T45	1	2022-11-13	11:45:00.0000000	Annual check - no issues	GGG1234	N
3	3	1	3	T40	3	2022-11-13	13:15:00.0000000	NULL	HHH1234	N
4	4	1	4	T45	2	2022-11-13	15:00:00.0000000	NULL	III1234	N
5	5	1	1	T48	1	2022-11-13	19:00:00.0000000	NULL	JJJ1234	N
6	6	2	5	T46	1	2022-11-13	10:45:00.0000000	Advised chew sticks	CCC1234	N
7	7	2	6	T47	1	2022-11-13	13:30:00.0000000	NULL	DDD1234	N

Figure 7 INSERT Queried

Retrieve Information using SQL Queries

MUST

1. Record information for each pet, including its type / species, approximate age (DoB not always known), comments.

```
-- Create tblPets
CREATE TABLE tblPets (
    petID INT IDENTITY(1,1) PRIMARY KEY,
    petName VARCHAR(25) NOT NULL,
    petTypeID CHAR(2) FOREIGN KEY REFERENCES tblPetTypes(petTypeID),
    ownerID INT FOREIGN KEY REFERENCES tblOwners(ownerID),
    petDOB DATE,
    petComments VARCHAR(1000)
);
```

Figure 8 Pet Table

The CREATE and INSERT scripts themselves allows you to record information for each pet, the **petTypeID** sets its species by referencing the *tblPetTypes* and both **petDOB** and **petComments** allow for the date of birth and further comment but aren't compulsory, an example of the 'INSERT' is on the previous page.

```
select * from tblPets
```

100 %

Results | Messages

	petID	petName	petTypeID	ownerID	petDOB	petComments
1	1	Ratty	RD	1	NULL	NULL
2	2	Carrots	RB	2	NULL	NULL
3	3	Pinchbeck	HR	3	NULL	NULL
4	4	Sleeky	CT	4	NULL	NULL
5	5	Rover	DG	5	NULL	NULL
6	6	Bugs	RB	6	NULL	NULL

Figure 9 Pet Table Queried

2. Enable one owner to be shown for multiple pets.

```
-- Create tblOwners
CREATE TABLE tblOwners (
    ownerID INT IDENTITY(1,1) PRIMARY KEY,
    ownerFName VARCHAR(25) NOT NULL,
    ownerLName VARCHAR(25) NOT NULL,
    address VARCHAR(500)
);

-- Create tblPets
CREATE TABLE tblPets (
    petID INT IDENTITY(1,1) PRIMARY KEY,
    petName VARCHAR(25) NOT NULL,
    petTypeID CHAR(2) FOREIGN KEY REFERENCES tblPetTypes(petTypeID),
    ownerID INT FOREIGN KEY REFERENCES tblOwners(ownerID),
    petDOB DATE,
    petComments VARCHAR(1000)
);
```

This is achieved by creating a 1-to-many relationship between *tblOwners* and *tblPets*, in the pets table we have an **ownerID** that is a foreign key referencing the owner's table.

This allows each pet to be associated with one owner while each owner can be associated with multiple pets.

Figure 10 Owner & Pet Tables

```
INSERT INTO tblPets (petName, petTypeID, ownerID, petDOB, petComments) VALUES
('Ratty The 2nd', 'RD', 1, '2022-11-1', 'Rattys little brother');

select * from tblPets where ownerID = '1';
```

91 %

Results | Messages

	petID	petName	petTypeID	ownerID	petDOB	petComments
1	1	Ratty	RD	1	NULL	NULL
2	11	Ratty The 2nd	RD	1	2022-11-01	Rattys little brother

Figure 11 INSERT 2nd Pet into Pet Table

3. Record staff information, identifying whether they are VETs or VET NURSEs.

```
-- Create tblStaffType
CREATE TABLE tblStaffType (
    staffTypeID INT IDENTITY(1,1) PRIMARY KEY,
    staffTypeDesc VARCHAR(10) NOT NULL
);

-- Create tblStaff
CREATE TABLE tblStaff (
    staffID INT IDENTITY(1,1) PRIMARY KEY,
    staffFName VARCHAR(25) NOT NULL,
    staffLName VARCHAR(25) NOT NULL,
    staffTypeID INT FOREIGN KEY REFERENCES tblStaffType(staffTypeID),
    lineManagerID INT FOREIGN KEY REFERENCES tblStaff(staffID)
);
```

This is handled by the *tblStaff* and *tblStaffType* tables where you can INSERT or record staff information, the staff table handles ID's, first and second names etc the **staffTypeID** foreign key in the staff table references the primary key of the staff type table, altogether this allows you to record staff information as well as identify and categorise staff by their type.

Figure 12 Staff and Staff Type Tables

```
select * from tblStaff join tblStaffType ON tblStaff.staffTypeID = tblStaffType.staffTypeID;
```

91 %

Results | Messages

	staffID	staffFName	staffLName	staffTypeID	lineManagerID	staffTypeID	staffTypeDesc
1	1	Gurkiran	Greenwell	1	NULL	1	VET
2	2	Sarah	Smith	2	NULL	2	VET NURSE

Figure 13 Staff & Staff Type JOIN

4. Record appointment information, showing pet and staff information as well as date and time. Allow space for outcome of appointment to be recorded.

```
-- Create tblAppointments
CREATE TABLE tblAppointments (
    appointmentID INT IDENTITY(1,1) PRIMARY KEY,
    staffID INT FOREIGN KEY REFERENCES tblStaff(staffID),
    petID INT FOREIGN KEY REFERENCES tblPets(petID),
    treatmentCode CHAR(3) FOREIGN KEY REFERENCES tblTreatments(treatmentCode),
    surgeryID INT NULL FOREIGN KEY REFERENCES tblSurgeries(surgeryID),
    date DATE NOT NULL,
    time TIME NOT NULL,
    outcome VARCHAR(100),
    invoiceNumber VARCHAR(10) UNIQUE NULL,
    paidStatus CHAR(1) CHECK (paidStatus IN ('Y', 'N')) NOT NULL DEFAULT 'N'
);
```

Figure 14 Appointment Table

To record appointment information, we have the appointments table where you can record or INSERT appointment information. The **staffID** and **petID** are both foreign keys referencing their own tables allowing you to associate a staff member and an animal for each appointment.

You also have the 'DATE' and 'TIME' columns for storing the date and time information as well as columns for the outcome of the treatment, invoice number and payment status.

```
select * from tblAppointments
```

91 %

Results | Messages

	appointmentID	staffID	petID	treatmentCode	surgeryID	date	time	outcome	invoiceNumber	paidStatus
1	1	1	1	T47	1	2022-11-13	09:00:00.0000000	biopsy sent	FFF1234	Y
2	2	1	2	T45	1	2022-11-13	11:45:00.0000000	Annual check - no issues	GGG1234	N
3	3	1	3	T40	3	2022-11-13	13:15:00.0000000	NULL	HHH1234	N
4	4	1	4	T45	2	2022-11-13	15:00:00.0000000	NULL	III1234	N
5	5	1	1	T48	1	2022-11-13	19:00:00.0000000	NULL	JJJ1234	N
6	6	2	5	T46	1	2022-11-13	10:45:00.0000000	Advised chew sticks	CCC1234	N
7	7	2	6	T47	1	2022-11-13	13:30:00.0000000	NULL	DDD1234	N

Figure 15 Appointment Table Queried

SHOULD

5. Show all appointments for today (including home visits) for a given staff member. Include time, pet, owner, staff, and treatment codes and cost.

```
select
    tblAppointments.date,
    tblAppointments.time,
    tblPets.petName,
    tblOwners.ownerFName,
    tblOwners.ownerLName,
    tblStaff.staffFName,
    tblStaff.staffLName,
    tblTreatments.treatmentCode,
    tblTreatments.treatmentCost,
    tblSurgeries.surgeryDetails
from tblAppointments
join tblStaff ON tblAppointments.staffID = tblStaff.staffID
join tblPets ON tblAppointments.petID = tblPets.petID
join tblOwners ON tblPets.ownerID = tblOwners.ownerID
join tblTreatments ON tblAppointments.treatmentCode = tblTreatments.treatmentCode
join tblSurgeries ON tblAppointments.surgeryID = tblSurgeries.surgeryID

where
    tblAppointments.date = '2022-11-13' AND
    tblAppointments.staffID = 1;
```

The 'SELECT' clause specifies the required columns I want to retrieve from the different tables and the 'FROM' clause selects the main table.

I used multiple 'JOIN' clauses to combine the rows from these tables based on the foreign keys from the appointments table, doing this we can join multiple tables together based on a related column/key.

Finally, the 'WHERE' clause will filter the results to only show appointments on the 13/11/2022 and where the staff ID is equal to '1'.

Figure 16 Appointment Query

	date	time	petName	ownerFName	ownerLName	staffFName	staffLName	treatmentCode	treatmentCost	surgeryDetails
1	2022-11-13	09:00:00.0000000	Ratty	Trevor	Tatler	Gurkiran	Greenwell	T47	33.00	Worthing Surgery Montague Place
2	2022-11-13	11:45:00.0000000	Carrots	Bettina	Bishop	Gurkiran	Greenwell	T45	45.00	Worthing Surgery Montague Place
3	2022-11-13	13:15:00.0000000	Pinchbeck	Sally	Smith	Gurkiran	Greenwell	T40	100.00	Home Visit
4	2022-11-13	15:00:00.0000000	Sleeky	Yolande	Yearling	Gurkiran	Greenwell	T45	45.00	Brighton Surgery Old Steine Place
5	2022-11-13	19:00:00.0000000	Ratty	Trevor	Tatler	Gurkiran	Greenwell	T48	150.00	Worthing Surgery Montague Place

Figure 17 Appointment Query Execution

6. Record payments made, including the invoice information, amount paid, method of payment etc. A payment can only be applied to one invoice (see Case Study). Generate a unique receipt number for each payment.

```
INSERT INTO tblPayments (ownerID, invoiceNumber, paymentAmount, paymentTypeID, paymentDate)
VALUES (1, 'FFF1234', 33, 2, '2022-11-13');
select * from tblPayments
```

90 %

Results | Messages

	paymentID	ownerID	invoiceNumber	paymentAmount	paymentTypeID	paymentDate
1	1	1	FFF1234	33.00	2	2022-11-13

Figure 18 INSERT into Payments Table

To record payments made including the required fields we can simply 'INSERT' into the *tlbPayments* table which has the fields already created.

The **invoiceNumber** foreign key in the payments table and **invoiceNumber** in the appointments table being 'UNIQUE' allows each payment to be applied to one invoice.

```
INSERT INTO tblPayments (ownerID, invoiceNumber, paymentAmount, paymentTypeID, paymentDate)
VALUES (1, 'FFF1234', 33, 2, '2022-11-13');
select * from tblPayments
```

90 %

Results | Messages

	paymentID	ownerID	invoiceNumber	paymentAmount	paymentTypeID	paymentDate
1	1	1	FFF1234	33.00	2	2022-11-13
2	4	1	FFF1234	33.00	2	2022-11-13

Figure 19 INSERT Multiple Payments

The auto-generated **paymentID** does allow for multiple payments to the same invoice but is recorded as a separate payment.

7. Show staff information, showing their line managers. Include staff who do not have line managers.

In my 'CREATE' statement we added a **lineManagerID** to the staff table but never gave it a description in the staff type table;

```
INSERT INTO tblStaffType (staffTypeDesc)
VALUES ('MANAGER');

select * from tblStaffType
```

90 %

Results | Messages

	staffTypeID	staffTypeDesc
1	1	VET
2	2	VET NURSE
3	1004	MANAGER

Figure 20 INSERT New Staff Type

Ideally I wanted **MANAGER** to be **staffTypeID** of 3 which is what I assumed would happen using 'IDENTITY(1,1)' however it turns out that when using this if a row is deleted or an insertion fails for whatever reason the identity value continues incrementing and does not ignore the fails or deletions but still guarantees uniqueness – to avoid this I'd have to alter or drop the table all together and remove the identity clause.

After that we can insert a new staff member to manage the staff and then add the new member of staff as Gurkirans manager;

```
INSERT INTO tblStaff (staffFName, staffLName, staffTypeID, lineManagerID)
VALUES ('Steven', 'Gerrard', 1004, NULL);

select * from tblStaff
```

90 %

Results | Messages

	staffID	staffFName	staffLName	staffTypeID	lineManagerID
1	1	Gurkiran	Greenwell	1	NULL
2	2	Sarah	Smith	2	NULL
3	1003	Steven	Gerrard	1004	NULL

Figure 21 INSERT New Line Manager

```
UPDATE tblStaff
SET lineManagerID = 1003
WHERE staffID = 1;
select * from tblStaff
```

90 %

Results | Messages

	staffID	staffFName	staffLName	staffTypeID	lineManagerID
1	1	Gurkiran	Greenwell	1	1003
2	2	Sarah	Smith	2	NULL
3	1003	Steven	Gerrard	1004	NULL

Figure 22 UPDATE Gurkirans Manager to New Line Manager

```
SELECT
    S1.staffID,
    S1.staffFName,
    S1.staffLName,
    S1.staffTypeID,
    S2.staffFName AS lineManagerFName,
    S2.staffLName AS lineManagerLName
FROM
    tblStaff AS S1
LEFT JOIN
    tblStaff AS S2 ON S1.lineManagerID = S2.staffID;
```

90 %

Results | Messages

	staffID	staffFName	staffLName	staffTypeID	lineManagerFName	lineManagerLName
1	1	Gurkiran	Greenwell	1	Steven	Gerrard
2	2	Sarah	Smith	2	NULL	NULL
3	1003	Steven	Gerrard	1004	NULL	NULL

Figure 23 Staff Table Self-Join Query

Now we specify the columns required, using aliases for the same table is useful when you want to compare data from the same table – we are essentially creating a staff table and a line manager table from the original staff table.

We use the 'FROM' clause to specify the main table which is the staff table.

Next, we *'LEFT JOIN' tblStaff* to itself known as a "self-join", this will return all the rows from the left table (S1 or staff table) and the matched rows from the right table (S2 or line manager table).

And finally, the condition for the join is that the **lineManagerID** in the staff/S1 table equals a **staffID** in the manager/S2 table – meaning the member of staff in S2 is the line manager of the member of staff in S1.

Note: a 'LEFT JOIN' is used to return all members of staff whether they have a line manager or not, if there is no manager then the result will be 'NULL'. *(SQL Server Tutorial*. 2021, *SQL joins*. Nd.)

8. Calculate the total owed by each owner (whether invoices are overdue or not).

```
SELECT o.ownerID, o.ownerFName, o.ownerLName, SUM(t.treatmentCost) AS totalOwed
FROM tblOwners o
LEFT JOIN tblPets p ON o.ownerID = p.ownerID
LEFT JOIN tblAppointments a ON p.petID = a.petID
LEFT JOIN tblTreatments t ON a.treatmentCode = t.treatmentCode
WHERE a.paidStatus = 'N'
GROUP BY o.ownerID, o.ownerFName, o.ownerLName;
```

90 %

Results | Messages

	ownerID	ownerFName	ownerLName	totalOwed
1	1	Trevor	Tatler	150.00
2	2	Bettina	Bishop	45.00
3	3	Sally	Smith	100.00
4	4	Yolande	Yearling	45.00
5	5	Paul	Preston	30.00
6	6	Praney	Patel	33.00

Figure 24 Total Owed Query

We 'SELECT' the necessary columns to be selected and then we use the 'SUM' aggregate function to calculate the total owed by each owner and assign it as a new column called **"totalOwed"**.

We once again use the 'LEFT JOIN' to join multiple tables together based on common keys, I used a 'LEFT JOIN' so I could return all rows from the owners table.

Finally, we use the 'WHERE' clause to filter the results based on the condition that the **paidStatus** is 'N' and then we use the 'GROUP BY' clause to group the results by the owners' details.

To test this, we have Trevor with **ownerID** of '1' who happens to have a paid appointment from the first appointment of the day which was a test with the treatment code of 'T47' and the cost of £33.

We can update the appointment table to return this **paidStatus** to 'N':

```
update tblAppointments
set paidStatus = 'N'
where petID = 1 AND paidStatus = 'Y';
select * from tblAppointments
```

90 %

Results | Messages

	appointmentID	staffID	petID	treatmentCode	surgeryID	date	time	outcome	invoiceNumber	paidStatus
1	1	1	1	T47	1	2022-11-13	09:00:00.0000000	biopsy sent	FFF1234	N
2	2	1	2	T45	1	2022-11-13	11:45:00.0000000	Annual check - no issues	GGG1234	N
3	3	1	3	T40	3	2022-11-13	13:15:00.0000000	NULL	HHH1234	N
4	4	1	4	T45	2	2022-11-13	15:00:00.0000000	NULL	III1234	N
5	5	1	1	T48	1	2022-11-13	19:00:00.0000000	NULL	JJJ1234	N
6	6	2	5	T46	1	2022-11-13	10:45:00.0000000	Advised chew sticks	CCC1234	N
7	7	2	6	T47	1	2022-11-13	13:30:00.0000000	NULL	DDD1234	N

Figure 21 Total Owed Test

We can then re-query the owners table to calculate the total owed by each owner and we should expect Trevor's total owed to increase from 150 to 183.

```
SELECT o.ownerID, o.ownerFName, o.ownerLName, SUM(t.treatmentCost) AS totalOwed
FROM tblOwners o
LEFT JOIN tblPets p ON o.ownerID = p.ownerID
LEFT JOIN tblAppointments a ON p.petID = a.petID
LEFT JOIN tblTreatments t ON a.treatmentCode = t.treatmentCode
WHERE a.paidStatus = 'N'
GROUP BY o.ownerID, o.ownerFName, o.ownerLName;
```

90 %

Results | Messages

	ownerID	ownerFName	ownerLName	totalOwed
1	1	Trevor	Tatler	183.00
2	2	Bettina	Bishop	45.00
3	3	Sally	Smith	100.00
4	4	Yolande	Yearling	45.00
5	5	Paul	Preston	30.00
6	6	Praney	Patel	33.00

Figure 26 Total Owed Test Result

COULD

9. Identify any unpaid invoices dated more than 28 days ago, showing the name and address of the responsible owner. Identify the owners that owe the most.

Let's start by creating some more data for the database, we will create two new owners with two new pets and two unpaid appointments from at least 28 days ago and then we can start querying it.

```
INSERT INTO tblOwners (ownerFName, ownerLName, address) VALUES
('Michael', 'Jackson', '123 something St'),
('Tito', 'Jackson', '456 something Ave');

select * from tblOwners
```

90 %

Results | Messages

	ownerID	ownerFName	ownerLName	address
1	1	Trevor	Tatler	NULL
2	2	Bettina	Bishop	NULL
3	3	Sally	Smith	NULL
4	4	Yolande	Yearling	NULL
5	5	Paul	Preston	NULL
6	6	Praney	Patel	NULL
7	1008	Michael	Jackson	123 something St
8	1009	Tito	Jackson	456 something Ave

Figure 27 INSERT Test Owners

```
INSERT INTO tblPets (petName, petTypeID, ownerID, petDOB, petComments) VALUES
('MJs Dog', 'DG', 1008, NULL, 'Michael Jacksons Dog!'),
('Titos Dog', 'DG', 1009, NULL, 'Tito Jacksons Dog!');

select * from tblPets
```

90 %

Results | Messages

	petID	petName	petTypeID	ownerID	petDOB	petComments
1	1	Ratty	RD	1	NULL	NULL
2	2	Carrots	RB	2	NULL	NULL
3	3	Pinchbeck	HR	3	NULL	NULL
4	4	Sleeky	CT	4	NULL	NULL
5	5	Rover	DG	5	NULL	NULL
6	6	Bugs	RB	6	NULL	NULL
7	11	Ratty The 2nd	RD	1	2022-11-01	Rattys little brother
8	1008	MJs Dog	DG	1008	NULL	Michael Jacksons Dog!
9	1009	Titos Dog	DG	1009	NULL	Tito Jacksons Dog!

Figure 28 INSERT Test Pets

```
INSERT INTO tblAppointments (staffID, petID, treatmentCode, surgeryID, date, time, outcome, invoiceNumber, paidStatus) VALUES
(1, 1008, 'T47', 1, '2022-09-13', '09:00', 'biopsy sent', 'MMM1234', 'N'),
(2, 1009, 'T45', 1, '2022-09-13', '11:45', 'Annual check - no issues', 'NNN1234', 'N');

select * from tblAppointments
```

90 %

Results | Messages

	appointmentID	staffID	petID	treatmentCode	surgeryID	date	time	outcome	invoiceNumber	paidStatus
1	1	1	1	T47	1	2022-11-13	09:00:00.0000000	biopsy sent	FFF1234	N
2	2	1	2	T45	1	2022-11-13	11:45:00.0000000	Annual check - no issues	GGG1234	N
3	3	1	3	T40	3	2022-11-13	13:15:00.0000000	NULL	HHH1234	N
4	4	1	4	T45	2	2022-11-13	15:00:00.0000000	NULL	III1234	N
5	5	1	1	T48	1	2022-11-13	19:00:00.0000000	NULL	JJJ1234	N
6	6	2	5	T46	1	2022-11-13	10:45:00.0000000	Advised chew sticks	CCC1234	N
7	7	2	6	T47	1	2022-11-13	13:30:00.0000000	NULL	DDD1234	N
8	1002	1	1008	T47	1	2022-09-13	09:00:00.0000000	biopsy sent	MMM1234	N
9	1003	2	1009	T45	1	2022-09-13	11:45:00.0000000	Annual check - no issues	NNN1234	N

Figure 29 INSERT Test Appointments

```
SELECT  O.ownerFName,   O.ownerLName, O.address,SUM(T.treatmentCost) AS totalDue
FROM tblOwners O
JOIN tblPets P ON O.ownerID = P.ownerID
JOIN tblAppointments A ON P.petID = A.petID
JOIN tblTreatments T ON A.treatmentCode = T.treatmentCode
WHERE
    A.paidStatus = 'N'
    AND A.date <= DATEADD(day, -28, '2022-11-13')
GROUP BY
    O.ownerFName, O.ownerLName, O.address;
```

90 %

Results | Messages

	ownerFName	ownerLName	address	totalDue
1	Michael	Jackson	123 something St	33.00
2	Tito	Jackson	456 something Ave	45.00

Figure 30 Query Test Appointments

Almost identical to the query from question 8 with a couple of exceptions, firstly we don't use a 'LEFT-JOIN' as we're not concerned about returning ALL owners just the ones with unpaid invoices over 28 days old and we also added an extra filter in the 'WHERE' clause using the 'DATEADD' function which filters the results to only the results that are -28 days from 13/11/2022.

(Dateadd (transact-SQL) - SQL server, 2022)

```
SELECT TOP 3 WITH TIES O.ownerFName, O.ownerLName, O.address, SUM(T.treatmentCost) AS TotalOwed
FROM tblOwners O
JOIN tblPets P ON O.ownerID = P.ownerID
JOIN tblAppointments A ON P.petID = A.petID
JOIN tblTreatments T ON A.treatmentCode = T.treatmentCode
WHERE A.paidStatus = 'N'
GROUP BY O.ownerFName, O.ownerLName, O.address
ORDER BY TotalOwed DESC;
```

90 %

Results | Messages

	ownerFName	ownerLName	address	TotalOwed
1	Trevor	Tatler	NULL	183.00
2	Sally	Smith	NULL	100.00
3	Tito	Jackson	456 something Ave	45.00
4	Bettina	Bishop	NULL	45.00
5	Yolande	Yearling	NULL	45.00

Figure 31 Owners That Owe the Most

Again, very similar queries except with the additions of 'TOP 3 WITH TIES' and the 'ORDER BY' clause, top 3 will like the name suggest return the top 3 results of the **TotalOwed** which includes ties as you can see from the image above - the order of the top 3 itself is determined by the 'ORDER BY TotalOwed DESC' clause. *(SQL Server select top by practical examples*, 2020)

10. Calculate the total amount owed to each surgery (consider how Home visits will be included).

```
SELECT S.surgeryDetails, SUM(T.treatmentCost) as TotalOwed
FROM tblSurgeries S
JOIN tblAppointments A ON S.surgeryID = A.surgeryID
JOIN tblTreatments T ON A.treatmentCode = T.treatmentCode
WHERE A.paidStatus = 'N'
GROUP BY S.surgeryDetails;
```

90 %

Results | Messages

	surgeryDetails	TotalOwed
1	Brighton Surgery Old Steine Place	45.00
2	Home Visit	100.00
3	Worthing Surgery Montague Place	369.00

Figure 32 Query Amount Owed to Each Surgery

Following the same general pattern we can select the **surgeryDetails** and the SUM of the **treatmentCost** as TotalOwed with our primary table being the surgeries table.

Next, we join the **surgeryID**'s from the surgery and appointment tables as well as the **treatmentCodes** from the treatment and appointment tables, 'WHERE' the paidStatus is equal to 'N' and finally we 'GROUP BY' the **surgeryDetails** itself rather than an owner.

Results | Messages

	appointmentID	staffID	petID	treatmentCode	surgeryID	date	time	outcome	invoiceNumber	paidStatus
1	1	1	1	T47	1	2022-11-13	09:00:00.0000000	biopsy sent	FFF1234	N
2	2	1	2	T45	1	2022-11-13	11:45:00.0000000	Annual check - no issues	GGG1234	N
3	3	1	3	T40	3	2022-11-13	13:15:00.0000000	NULL	HHH1234	N
4	4	1	4	T45	2	2022-11-13	15:00:00.0000000	NULL	III1234	N
5	5	1	1	T48	1	2022-11-13	19:00:00.0000000	NULL	JJJ1234	N
6	6	2	5	T46	1	2022-11-13	10:45:00.0000000	Advised chew sticks	CCC1234	N
7	7	2	6	T47	1	2022-11-13	13:30:00.0000000	NULL	DDD1234	N
8	1002	1	1008	T47	1	2022-09-13	09:00:00.0000000	biopsy sent	MMM1234	N
9	1003	2	1009	T45	1	2022-09-13	11:45:00.0000000	Annual check - no issues	NNN1234	N

Figure 33 Appointment Table

To test this, most appointments are in the Worthing Surgery or **surgeryID** of '1' except one appointment at the Brighton Surgery, lets change this one appointment to the Worthing Surgery.

```
UPDATE tblAppointments
SET surgeryID = 1
WHERE surgeryID = 2;
select * from tblAppointments
```

90 %

Results | Messages

	appointmentID	staffID	petID	treatmentCode	surgeryID	date	time	outcome	invoiceNumber	paidStatus
1	1	1	1	T47	1	2022-11-13	09:00:00.0000000	biopsy sent	FFF1234	N
2	2	1	2	T45	1	2022-11-13	11:45:00.0000000	Annual check - no issues	GGG1234	N
3	3	1	3	T40	3	2022-11-13	13:15:00.0000000	NULL	HHH1234	N
4	4	1	4	T45	1	2022-11-13	15:00:00.0000000	NULL	III1234	N
5	5	1	1	T48	1	2022-11-13	19:00:00.0000000	NULL	JJJ1234	N
6	6	2	5	T46	1	2022-11-13	10:45:00.0000000	Advised chew sticks	CCC1234	N
7	7	2	6	T47	1	2022-11-13	13:30:00.0000000	NULL	DDD1234	N
8	1002	1	1008	T47	1	2022-09-13	09:00:00.0000000	biopsy sent	MMM1234	N
9	1003	2	1009	T45	1	2022-09-13	11:45:00.0000000	Annual check - no issues	NNN1234	N

Figure 34 UPDATE Appointment Table

Once changed we can then re-query the appointment table to check the script.

```
SELECT S.surgeryDetails, SUM(T.treatmentCost) as TotalOwed
FROM tblSurgeries S
JOIN tblAppointments A ON S.surgeryID = A.surgeryID
JOIN tblTreatments T ON A.treatmentCode = T.treatmentCode
WHERE A.paidStatus = 'N'
GROUP BY S.surgeryDetails;
```

90 %

Results | Messages

	surgeryDetails	TotalOwed
1	Home Visit	100.00
2	Worthing Surgery Montague Place	414.00

Figure 35 Re-Query Amount Owed to Each Surgery

As you can see the £45 that was owed to the Brighton Surgery has now been transferred over to the Worthing Surgery.

11. Find the average income generated by each staff member per month (ignore unpaid invoices).

Firstly, lets insert some data into the appointments table for my two staff members for the previous 2 months and update the current appointment tables **paidStatus**'s to 'Y' or *paid*.

Note: We will leave one appointment as 'N' or *unpaid* for testing purposes.

```
INSERT INTO tblAppointments
(staffID, petID, treatmentCode, surgeryID, date, time, outcome, invoiceNumber, paidStatus)
VALUES
(1, 1, 'T40', 1, '2022-10-01', '09:00', 'TEST', 'TEST1001', 'Y'),
(1, 2, 'T47', 1, '2022-10-15', '11:45', 'TEST', 'TEST1002', 'Y'),
(1, 1, 'T40', 1, '2022-09-01', '09:00', 'TEST', 'TEST1003', 'Y'),
(1, 2, 'T47', 1, '2022-09-15', '11:45', 'TEST', 'TEST1004', 'Y');

INSERT INTO tblAppointments
(staffID, petID, treatmentCode, surgeryID, date, time, outcome, invoiceNumber, paidStatus)
VALUES
(2, 3, 'T40', 1, '2022-10-02', '10:00', 'TEST', 'TEST2001', 'Y'),
(2, 4, 'T47', 1, '2022-10-16', '12:45', 'TEST', 'TEST2002', 'Y'),
(2, 3, 'T40', 1, '2022-09-02', '10:00', 'TEST', 'TEST2003', 'Y'),
(2, 4, 'T47', 1, '2022-09-16', '12:45', 'TEST', 'TEST2004', 'Y');

UPDATE tblAppointments
SET paidStatus = 'Y';
UPDATE tblAppointments
SET paidStatus = 'N'
WHERE invoiceNumber = 'FFF1234';

select * from tblAppointments
```

90 %

Results | Messages

	appointmentID	staffID	petID	treatmentCode	surgeryID	date	time	outcome	invoiceNumber	paidStatus
1	1	1	1	T47	1	2022-11-13	09:00:00.0000000	biopsy sent	FFF1234	N
2	2	1	2	T45	1	2022-11-13	11:45:00.0000000	Annual check - no issues	GGG1234	Y
3	3	1	3	T40	3	2022-11-13	13:15:00.0000000	NULL	HHH1234	Y
4	4	1	4	T45	1	2022-11-13	15:00:00.0000000	NULL	III1234	Y
5	5	1	1	T48	1	2022-11-13	19:00:00.0000000	NULL	JJJ1234	Y
6	6	2	5	T46	1	2022-11-13	10:45:00.0000000	Advised chew sticks	CCC1234	Y
7	7	2	6	T47	1	2022-11-13	13:30:00.0000000	NULL	DDD1234	Y
8	1002	1	1008	T47	1	2022-09-13	09:00:00.0000000	biopsy sent	MMM1234	Y
9	1003	2	1009	T45	1	2022-09-13	11:45:00.0000000	Annual check - no issues	NNN1234	Y
10	1012	1	1	T40	1	2022-10-01	09:00:00.0000000	TEST	TEST1001	Y
11	1013	1	2	T47	1	2022-10-15	11:45:00.0000000	TEST	TEST1002	Y
12	1014	1	1	T40	1	2022-09-01	09:00:00.0000000	TEST	TEST1003	Y
13	1015	1	2	T47	1	2022-09-15	11:45:00.0000000	TEST	TEST1004	Y
14	1016	2	3	T40	1	2022-10-02	10:00:00.0000000	TEST	TEST2001	Y
15	1017	2	4	T47	1	2022-10-16	12:45:00.0000000	TEST	TEST2002	Y
16	1018	2	3	T40	1	2022-09-02	10:00:00.0000000	TEST	TEST2003	Y
17	1019	2	4	T47	1	2022-09-16	12:45:00.0000000	TEST	TEST2004	Y

Figure 36 INSERT New Test Data

```
SELECT S.staffFName, S.staffLName, A.staffID, MONTH(A.date) AS Month, YEAR(A.date) AS Year, AVG(T.treatmentCost) AS AverageIncomePerMonth
FROM tblAppointments A
JOIN tblTreatments T ON A.treatmentCode = T.treatmentCode
JOIN tblStaff S ON A.staffID = S.staffID
WHERE A.paidStatus = 'Y'
GROUP BY S.staffFName, S.staffLName, A.staffID, MONTH(A.date), YEAR(A.date);
```

90 %

Results | Messages

	staffFName	staffLName	staffID	Month	Year	AverageIncomePerMonth
1	Gurkiran	Greenwell	1	9	2022	55.333333
2	Gurkiran	Greenwell	1	10	2022	66.500000
3	Gurkiran	Greenwell	1	11	2022	85.000000
4	Sarah	Smith	2	9	2022	59.333333
5	Sarah	Smith	2	10	2022	66.500000
6	Sarah	Smith	2	11	2022	31.500000

Figure 37 Query Average Income Per Month Per Employee

In September Sarah (**staffID** = 2) had 3 appointments with the treatment codes of T40, T45 and T47 with their respective costs being 100, 45 and 33 – (100+45+33)/3 = 59.33*.

In November Sarah had 2 appointments with the treatment codes of T46 and T47 with their respective costs being 30 and 33 – (30+33)/2 = 31.5.

In September Gurkiran (**staffID** = 1) had 3 appointments with the treatment codes of T40, T47 and T47 with their respective costs being 100, 33 and 33 – (100+33+33)/3 = 55.33*.

In November Gurkiran had 5 appointments with the treatment codes of T47, T45, T45, T40 and T48 with their respective costs being 33, 45, 45, 100 and 150 but we have to take into account 1 of the appointments **paidStatus** is set to 'N' or *unpaid* which should not return in the SQL query, that appointment had the treatment code of T47 – (45+45+100+150)/4 = 85.

Note: If you want the total income per month per employee simply replace the **"AVG(T.treatmentCost) As AverageIncomePerMonth"** to **"SUM(T.treatmentCost) As TotalIncomePerMonth**.

Implementation of DBMS Functionality

WOULD BE NICE

12. Create a stored procedure to input new pet information, including associating them with an existing or new owner (given the owner's name).

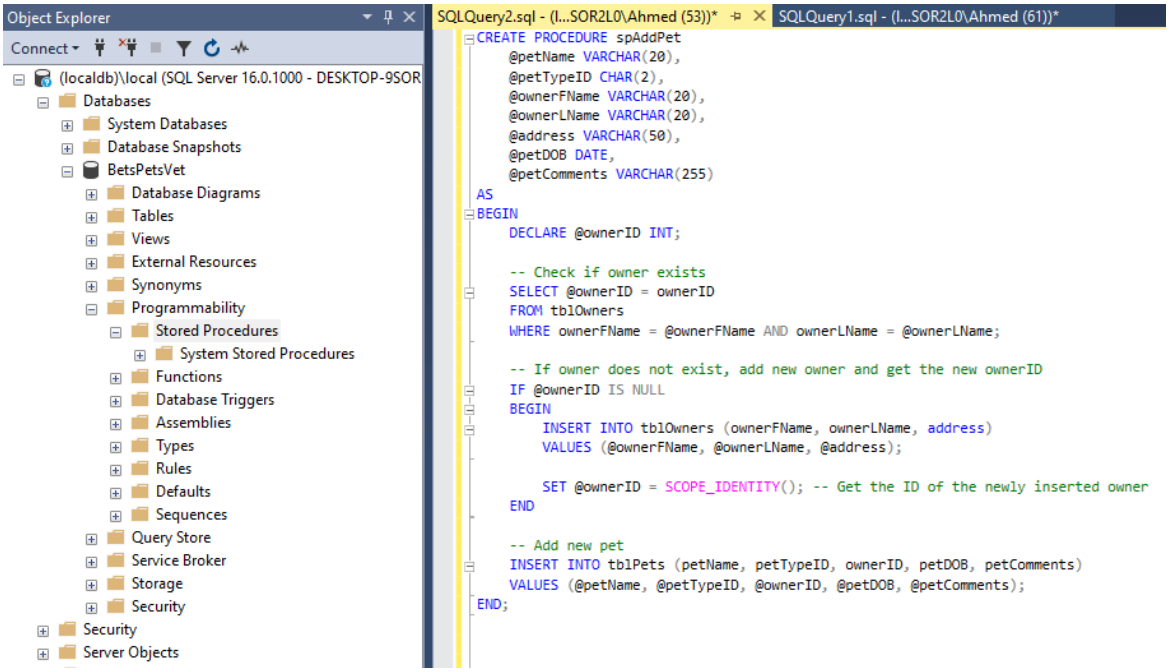


Figure 38 Creating a Stored Procedure

To create a stored procedure, we navigate to the Stored Procedure directory and right click on the "New Stored Procedure" option, it will automatically generate a template procedure in a new window for you.

To begin with we create a new procedure, name it, and then define the parameters the stored procedure will take.

'AS BEGIN' is the start of the stored procedures block of code where we declare **@ownerID** as an INT, we then check if owner already exists by comparing our **@ownerID**, **@ownerFName** and **@ownerLName** to their counterparts in the owner's table – if owner exists, the **ownerID** is stored in the **@ownerID** variable and we skip to the final 'INSERT' and add the new pet to the **@ownerID**.

The following block of code runs if the **@ownerID** is *NULL* or doesn't exist and simply inserts a new owner into the owner table and using the 'SCOPE_IDENTITY()' function we can retrieve the last identity value that was inserted (as it is auto-generated) within the current scope and assign it to the **@ownerID** variable - once the owner is added we can proceed to the final 'INSERT' and add the new pet as well.

(Assaf, W.D. no date *Create a stored procedure - SQL server.*

Medewar, S. and Thul, B. 2010 *How to use scope_identity to retrieve the last ID that was inserted.*

VanMSFT. 2023 *Scope_identity (transact-SQL) - SQL server)*

To test our new stored procedure, we can use the 'EXEC' command followed by the stored procedures name to execute it with the required parameters and then we can check the owners and pets table to see if they have been updated.

```
EXEC spAddPet 'testPet', 'DG', 'Bobby', 'Firminho', '123 test St', '2021-03-04', 'Bobbys Dog';

SELECT * FROM tblPets;
SELECT * FROM tblOwners;
```

90 %

Results | Messages

	petID	petName	petTypeID	ownerID	petDOB	petComments
3	3	Pinchbeck	HR	3	NULL	NULL
4	4	Sleeky	CT	4	NULL	NULL
5	5	Rover	DG	5	NULL	NULL
6	6	Bugs	RB	6	NULL	NULL
7	11	Ratty The 2nd	RD	1	2022-11-01	Rattys little brother
8	1008	MJs Dog	DG	1008	NULL	Michael Jacksons Dog!
9	1009	Titos Dog	DG	1009	NULL	Tito Jacksons Dog!
10	1010	testPet	DG	1010	2021-03-04	Bobbys Dog

	ownerID	ownerFName	ownerLName	address
1	1	Trevor	Tatler	NULL
2	2	Bettina	Bishop	NULL
3	3	Sally	Smith	NULL
4	4	Yolande	Yearling	NULL
5	5	Paul	Preston	NULL
6	6	Praney	Patel	NULL
7	1008	Michael	Jackson	123 s...
8	1009	Tito	Jackson	456 s...
9	1010	Bobby	Firminho	123 t...

Figure 39 Testing a Stored Procedure

13. Create a trigger that will check how much a given owner owes when they make an appointment. If there are any unpaid invoices more than 28 days old, do not make the booking.

```
CREATE TRIGGER trg_check_unpaid_invoices
ON tblAppointments
AFTER INSERT
AS
BEGIN
    DECLARE @ownerID INT;
    DECLARE @petID INT;

    -- Get the petID from the inserted row
    SELECT @petID = petID FROM inserted;

    -- Get the ownerID related to the pet from tblPets
    SELECT @ownerID = ownerID FROM tblPets WHERE petID = @petID;

    -- Check if the owner has any unpaid appointments that are more than 28 days old
    IF EXISTS (
        SELECT 1
        FROM tblAppointments A
        JOIN tblPets P ON A.petID = P.petID
        WHERE P.ownerID = @ownerID
        AND A.paidStatus = 'N'
        AND A.date < DATEADD(DAY, -28, '2022-11-13')
    )
    BEGIN
        -- If appointment exist, rollback the transaction to prevent the booking
        ROLLBACK TRANSACTION;
        THROW 2147483647, 'Owner has unpaid appointments more than 28 days old, DO NOT BOOK', 1;
    END
END;
```

Figure 40 Creating a Trigger

To begin with 'CREATE TRIGGER' followed by the name, the table you want the trigger to execute on, and declare it to execute 'AFTER INSERT'.

Again 'AS BEGIN' starts the block of code for the trigger, we declare 2 INT variables - @**ownerID** and @**petID** and then we assign these variables to the newly inserted **ownerID** and **petID.**

Next we check if the owner has any unpaid appointments that are -28 days from the current date of the database (13/11/2022), this is very similar to the script from question 9 with the the difference of 'SELECT 1' which will select any unpaid appointment and run the ROLLBACK.

If an unpaid appointment exists the 'ROLLBACK TRANSACTION' begins and 'THROWS' an exception/error message.

(SQL server, no date. *Create trigger (transact-SQL), Gullezn, G. 2012, Trigger SQL server (see if a hotelroom is already booked))*

Now to test the trigger we can set 2 appointments that are 28 days or older to the **paidStatus** of 'N' or *unpaid* and then we can try to insert these pets into new appointments, and we can expect to see the error message.

```
UPDATE tblAppointments
SET paidStatus = 'N'
WHERE petID IN (1008, 1009);
select * from tblAppointments


-- TEST TRIGGER PETID 1008
INSERT INTO tblAppointments (staffID, petID, treatmentCode, surgeryID, date, time, outcome, invoiceNumber, paidStatus)
VALUES (1, 1008, 'T47', 1, '2022-11-13', '09:00', 'biopsy sent', 'MMM1235', 'N');

-- TEST TRIGGER PETID 1009
INSERT INTO tblAppointments (staffID, petID, treatmentCode, surgeryID, date, time, outcome, invoiceNumber, paidStatus)
VALUES (2, 1009, 'T45', 1, '2022-11-13', '11:45', 'Annual check - no issues', 'NNN1235', 'N');
```

```
%
Messages
Msg 2147483647, Level 16, State 1, Procedure trg_check_unpaid_invoices, Line 27 [Batch Start Line 38]
Owner has unpaid appointments more than 28 days old, DO NOT BOOK
```

Figure 41 Testing the Trigger

14. Create an app capable of running the queries.

I created a GUI which would have the ability to add, remove, update, and search like a basic CRUD application, unfortunately I believe firewall issues stopped me from connecting to the database.

```
package com.example.betspetsvetgui;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.stage.Stage;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.ResultSet;

import java.io.IOException;

3 usages  new *
public class HelloApplication extends Application {
    3 usages
    private static Connection con;
    new *
    @Override
    public void start(Stage stage) throws IOException {
        FXMLLoader fxmlLoader = new FXMLLoader(HelloApplicat
        Scene scene = new Scene(fxmlLoader.load(),  v: 1200,
        stage.setTitle("Hello!");
        stage.setScene(scene);
        stage.show();
    }
```

Figure 42 SQL Imports

To begin with I imported all required statements to allow JavaFX to work with SQL Server.

As well as declaring “con” a **Connection** object and the ‘start’ method for the application.

Next the **connectDatabase** method was a template method pulled from SQL documentation that is meant to first load the Microsoft SQL Server drivers using ‘Class.forName’.

After we define the connection URLs, I was using localdb with a server name of ‘local’ with the database called ‘BetsPetsVet’

Finally, we attempt a connection and output to console on successful connection.

getConnection is the getter method that returns the con Connection object.

```
no usages  new *
public static void main(String[] args) {
    connectDatabase(); launch();
}
1 usage  new *
private static void connectDatabase() {
    try {
        // Load Microsoft SQL Server JDBC driver
        Class.forName( className: "com.microsoft.sqlserver.jdbc.SQLServerDriver");

        // Define connection url
        String connectionUrl = "jdbc:sqlserver://localhost;instance=(localdb)\\local;databaseName=BetsPetsVet;integratedSecurity=true";

        // Establish the connection
        con = DriverManager.getConnection(connectionUrl);

        // If connection is successful
        if (con != null) {
            System.out.println("Connected to the database!");
        }

    } catch (SQLException | ClassNotFoundException ex) {
        ex.printStackTrace();
    }
}

1 usage  new *
public static Connection getConnection() {
    return con;
}
```

Figure 43 connectDatabase Method

Using SceneBuilder I set up the user input fields and declare a ListView for output display and then referenced them in the FXML file and my controller class.

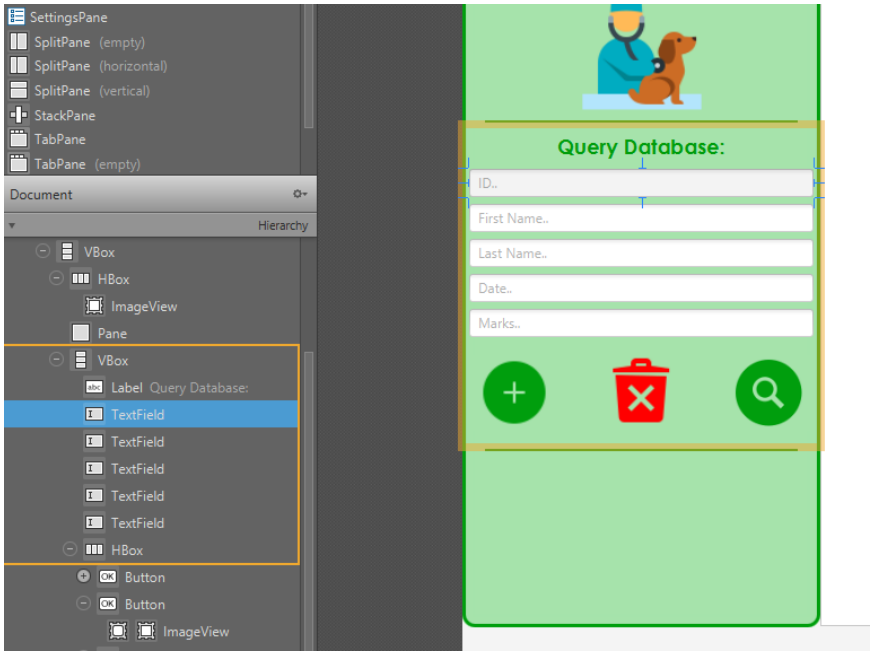


Figure 44 GUI Inputs

```
1 usage
@FXML
private Button searchButton;

2 usages
@FXML
private TextField idField;

2 usages
@FXML
private TextField FirstNameField;

2 usages
@FXML
private TextField LastNameField;

2 usages
@FXML
private TextField DateField;

3 usages
@FXML
private ListView listView;
```

Figure 45 FXML Inputs

Finally, to try and put it all together I added an EventListener to my Search button that would run my 'handleSearch' method when clicking it.

```
1 usage  new *
@FXML
protected void handleSearch() {
    Connection con = HelloApplication.getConnection();

    String query = "SELECT * FROM tblAppointments WHERE staffID = ? AND staffFName = ? AND staffLName = ? AND date = ?";
    try {
        PreparedStatement pstmt = con.prepareStatement(query);
        pstmt.setString( parameterIndex: 1, idField.getText());
        pstmt.setString( parameterIndex: 2, FirstNameField.getText());
        pstmt.setString( parameterIndex: 3, LastNameField.getText());
        pstmt.setDate( parameterIndex: 4, java.sql.Date.valueOf(DateField.getText()));
        ResultSet rs = pstmt.executeQuery();

        // clear ListView before adding new results
        listView.getItems().clear();

        while (rs.next()) {
            // concatenating results into one string and adding it to the ListView
            listView.getItems().add(rs.getString( columnLabel: "staffID") + " " +
                    rs.getString( columnLabel: "staffFName") + " " +
                    rs.getString( columnLabel: "staffLName") + " " +
                    rs.getDate( columnLabel: "date"));
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Figure 46 handleSearch Method

The problem I have is when I launch the application the first thing it tries to do before even opening the application window is to connect to the database, this is the error I get when launching.

```
"C:\Program Files\Java\jdk-19\bin\java.exe" ...
com.microsoft.sqlserver.jdbc.SQLServerException Create breakpoint : The TCP/IP connection to the host localhost, port 1433 has failed. Error: "Connection refused: no further information.
Verify the connection properties. Make sure that an instance of SQL Server is running on the host and accepting TCP/IP connections at the port. Make sure that TCP connections to the port are not blocked by a firewall.".
```

Figure 47 Connection Errors

I tried enabling TCP/IP through the SQL Server Configuration Manager and set the Dynamic port to zero with the same error.

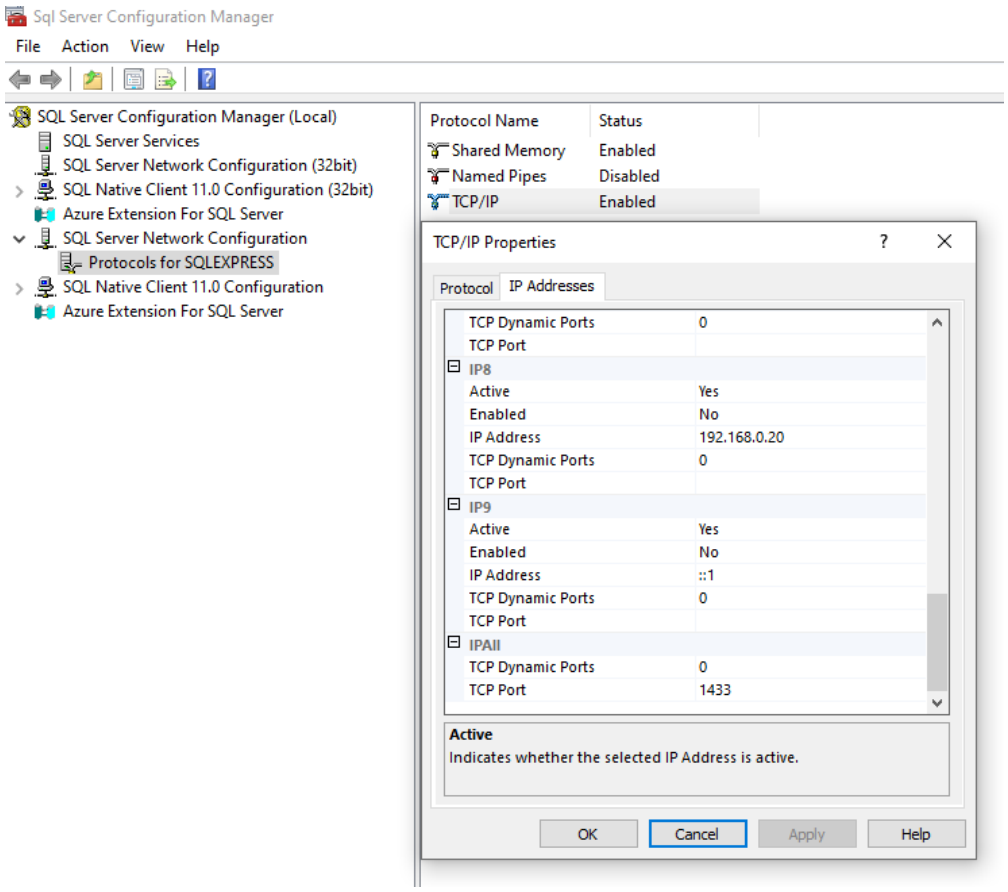


Figure 48 Attempt to Open SQL Server Ports

I've tried disabling my Firewall as well as Windows Defender Firewall to no success, I've also tried opening TCP/IP port 1433 (the default port for SQL Server) through registry keys which doesn't seem to help.

Business Justification

The implemented features, procedures and triggers in the database will improve the operational efficiency of 'Bets Pets Vets' which will keep its clients happy as well as meeting and maintaining business objectives.

To begin with the stored procedure for adding new pet information and then associating it with an existing or new owner provides a smooth and efficient way of updating and recording records. It ensures that any new pet/owner information is recorded accurately and reduces the chance of human error, potential data loss and re-enforces referential integrity.

The trigger for checking unpaid invoices before making an appointment also reduces the chance of human error by not booking appointments when money is due, for the business this will help sustain cash flow and reduce debts to clients.

Potential improvements

- → Create a trigger to remind clients a day or 2 prior to appointments by SMS or email – this could help avoid missing appointments and re-scheduling but would need to integrate some form of communication.
- → Create a trigger to remind clients of overdue bills, again by SMS/email or call – this could be automated to be sent out if no payment has been made after 2 weeks of appointment.
- → Creating an inventory/medicine table to keep track of stock and supplies, this would include item name, quantity, expiration date and supplier information – could also create a reorder trigger on low supplied items either with automatic re-order or notifications.
- → Create all new stored procedures and triggers to manage and track inventory i.e., adding new items etc.
- → Report and Analytics feature on both stock, staff, and treatments, potentially add success rates to treatments, staff proficiency, how fast stock goes etc.

In conclusion, the features implemented in this database will not only greatly increase the efficiency of administrative tasks at the vet but also help avoid the risk of human error while also improving customer service quality.

References

Assaf, W.D. (no date) Create a stored procedure - SQL server, Create a Stored Procedure - SQL Server | Microsoft Learn. Available at: https://learn.microsoft.com/en-us/sql/relational-databases/stored-procedures/create-a-stored-procedure?view=sql-server-ver16.

Dateadd (transact-SQL) - SQL server (2022) SQL Server | Microsoft Learn. Available at: https://learn.microsoft.com/en-us/sql/t-sql/functions/dateadd-transact-sql?view=sql-server-ver16.

Gullezn, G. (2012) Trigger SQL server (see if a hotelroom is already booked), Stack Overflow. Available at: https://stackoverflow.com/questions/35014987/trigger-sql-server-see-if-a-hotelroom-is-already-booked.

Medewar, S. and Thul, B. (2010) How to use scope_identity to retrieve the last ID that was inserted, Stack Overflow. Available at: https://stackoverflow.com/questions/8986278/how-to-use-scope-identity-to-retrieve-the-last-id-that-was-inserted.

SQL server (no date) Create trigger (transact-SQL) - SQL Server | Microsoft Learn. Available at: https://learn.microsoft.com/en-us/sql/t-sql/statements/create-trigger-transact-sql?view=sql-server-ver16.

SQL Server select top by practical examples (2020) SQL Server Tutorial. Available at: https://www.sqlservertutorial.net/sql-server-basics/sql-server-select-top/.

SQL Server Tutorial. (2021) SQL Server Self join by practical examples Available at: https://www.sqlservertutorial.net/sql-server-basics/sql-server-self-join/.

SQL joins. (No date) Available at: https://www.w3schools.com/sql/sql_join.asp.

VanMSFT (2023) Scope_identity (transact-SQL) - SQL server, SQL Server | Microsoft Learn. Available at: https://learn.microsoft.com/en-us/sql/t-sql/functions/scope-identity-transact-sql?redirectedfrom=MSDN&view=sql-server-ver16.