Linnéuniversitetet
Kalmar Växjö

Report

Database Theory

Final Project

Författare: Ruth Dirnfeld
Helena Tevar
Handledare: Maria Ullan
Examinator: Illir Jusufi
Termin: HT18
Ämne: Database Theory
Kurskod: 2DV513

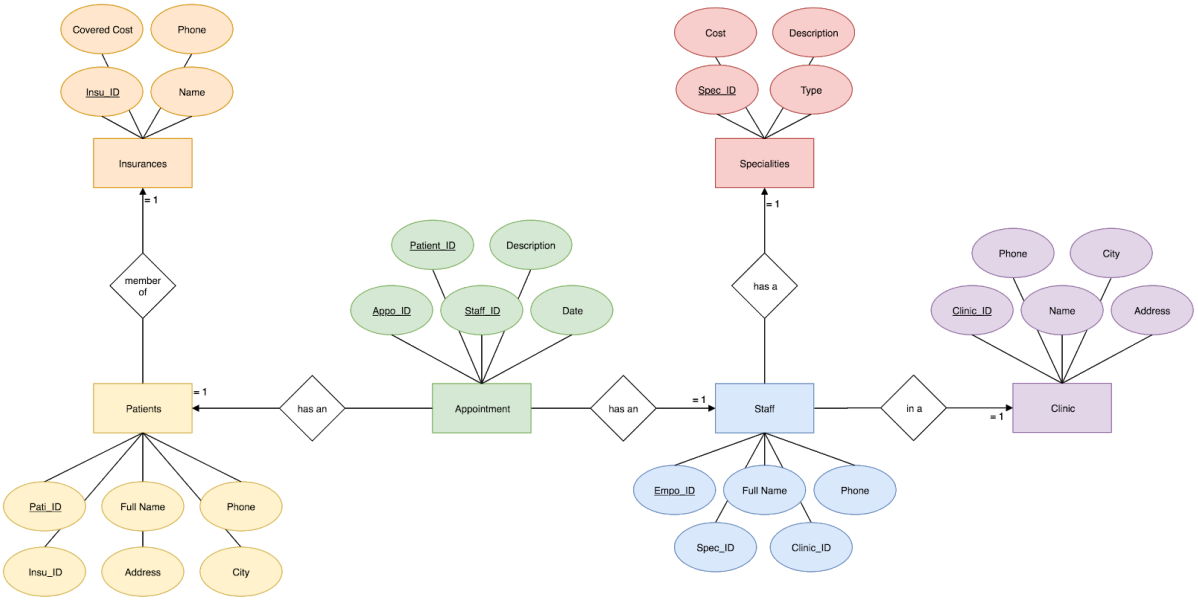
Innehållsförteckning

Idea

The project idea was to create a database of a medical clinic or group of clinic to be used by their staff. For instance, administrative staff, to manage patients, medical staff and the appointments.
The application is complete “as is” and is suited for all staff because the aim is to show the results of the useful working queries in a graphical user interface (gui) and not to add/update/delete any records. (Since that was not a requirement anyways)
The application was developed in the programming language C# and instructions on how to execute the application with all required prerequisites are mentioned at the end of this document.

Logical Model

In order to have a clear idea of the tables to be created, we made a E/R diagram as shown below.



Our databases handle three main entities, patients, staff and appointments. All the entities represent real life objects, such as appointments and clinics so the diagram is faithful.
Two entities had a special treatment, as Insurances and Specialities. Those two entities may have been part of the attributes of Patients and Staff, but it would create some problems of redundancy and other anomalies, that has been eliminated by decomposing those relations into different entities. Thanks to this we can ensure avoiding as much redundancy as possible (or desired). After the addition of those two entities, we see no need to add more entities in order to keep the schema simple enough.

Equally important the last of the design principles refer to the creation of right relationships. We agree that the relationships from our diagram are faithful, simple and not redundant, so there is no reason to think that they are not right.[1]

Design in SQL

By translating our E/R diagram into relations we get the set of relations below:

Patients(PatientID, Name, Phone, Address, City, InsuranceID)
Insurances(InsuranceID, Name, Phone, CoveredCost)
Staff (StaffID, Name, Phone, SpecialityID, ClinicID)
Specialities(SpecialityID, Title, Description, Cost)
Clinics(ClinicID, Name, Address, City, Phone)
Consults(ConsultID, PatientID, EmployeeID, Date, Notes)

Important to mention that Appointments was changed to Consults because the simplicity of the name. Consults is shorter and would be a help in the implementation.

As can be seen in our set of relations, it fulfils the requirement to be BCNF, for instance the relation Staff, its key determine all the other attributes of the relation.
The set of relations are transitive, so for instance looking at Consults, the closure of ConsultID, PatientID, EmployeeID can reach all the attributes of all the relations by using transitivity in the set of relations. This model is straightforward in order to avoid useless redundancies, update anomalies and deletion anomalies.

Tables creation

The code below was used to create the group of tables required for this project using SQL Server in Visual Studio 2017.

Patients

```
CREATE TABLE [dbo].[Patients] (
    [PatientID]   INT IDENTITY (1, 1) NOT NULL,
    [Name]        NVARCHAR (50) NOT NULL,
    [Address]     NVARCHAR (50) NOT NULL,
    [City]        NVARCHAR (50) NOT NULL,
    [Phone]       NVARCHAR (50) NOT NULL,
    [InsuranceID] INT           NOT NULL,
    PRIMARY KEY CLUSTERED ([PatientID] ASC)
);
```

Insurances

```
CREATE TABLE [dbo].[Insurances] (
    [InsuranceID] INT IDENTITY (1, 1) NOT NULL,
```

[1] Garcia-Molina, Ullman, Widom. *Database Systems The complete Book*. 2nd Edition. p. 140

```
    [Name]        NVARCHAR (50) NOT NULL,
    [Phone]       NVARCHAR (50) NOT NULL,
    [CoveredCost] INT           NOT NULL,
    PRIMARY KEY CLUSTERED ([InsuranceID] ASC)
);
```

Staff

```
CREATE TABLE [dbo].[Staff] (
    [StaffID]      INT IDENTITY (1, 1) NOT NULL,
    [Name]         NVARCHAR (50) NOT NULL,
    [Phone]        NVARCHAR (50) NOT NULL,
    [ClinicID]     INT           NOT NULL,
    [SpecialityID] INT           NOT NULL,
    PRIMARY KEY CLUSTERED ([StaffID] ASC)
);
```

Specialities

```
CREATE TABLE [dbo].[Specialities] (
    [SpecialityID] INT IDENTITY (1, 1) NOT NULL,
    [Title]        NVARCHAR (50) NOT NULL,
    [Description]  NVARCHAR (50) NOT NULL,
    [Cost]         INT           NOT NULL,
    PRIMARY KEY CLUSTERED ([SpecialityID] ASC)
);
```

Clinics

```
CREATE TABLE [dbo].[Clinics] (
    [ClinicID] INT IDENTITY (1, 1) NOT NULL,
    [Name]     NVARCHAR (50) NOT NULL,
    [Address]  NVARCHAR (50) NOT NULL,
    [Phone]    NVARCHAR (50) NOT NULL,
    [City]     NVARCHAR (50) NOT NULL,
    PRIMARY KEY CLUSTERED ([ClinicID] ASC)
);
```

Consults

```
CREATE TABLE [dbo].[Consults] (
    [ConsultsID] INT IDENTITY (1, 1) NOT NULL,
    [PatientID]  INT           NOT NULL,
    [StaffID]    INT           NOT NULL,
    [Date]       NCHAR (10)    NOT NULL,
    [Notes]      NVARCHAR (50) NOT NULL,
    PRIMARY KEY CLUSTERED ([ConsultsID] ASC)
```

```
);
```

SQL Queries

We are going to omit simple queries used to fill the grids in our windows form. All of them look like below, so we are going to assume that those queries are already self explanatories.

```
SELECT *
FROM table
```

In order to make this report more readable, we added a table of what expected result we wanted from the query created. The results are not literal, but orientative.

Patients and Insurances

```
SELECT Patients.Name, Patients.Phone, Patients.Address,
Patients.City, Insurances.Name
FROM Patients, Insurances
WHERE Patients.InsuranceID = Insurances.InsuranceID
```

Expected result:

Name	Phone	Address	City	Insurance
Ana	111111	Storgatan	Växjö	Vidas
John	22222	Vikingatan	Alvesta	Life

Staff, specialities and clinics

```
SELECT Staff.Name, Staff.Phone, Specialities.Title,
Clinics.Name
FROM Staff, Specialities, Clinics
WHERE Staff.SpecialityID = Specialities.SpecialityID
AND Staff.ClinicID = Clinics.ClinicID
```

Expected result:

Name	Phone	Speciality	Clinic
Tomas	33333	Nurse	Växjö Clinic
Marta	44444	Family Doctor	Alvesta Clinic

Consults, the readable table

```
SELECT Consults.ConsultID, Patients.Name, Specialities.Title,
Clinics.Name, Consults.Date, Consults.Notes
```

```
FROM Consults, Patients, Specialities, Clinics, Staff
WHERE Consults.PatientsID = Patients.PatientsID
AND Consults.StaffID = Staff.StaffID
AND Staff.ClinicID = Clinics.ClinicID
```

Expected result:

ConsultID	Patient Name	Speciality	Clinic	Date	Notes
1	Isa K	Nurse	Växjö C	01/01/19	Meeting
2	James L	Dermatologist	Alvesta C	02/01/19	Checking

Staff and a patient

```
SELECT Staff.Name
FROM Staff
WHERE Staff.StaffID IN (
    SELECT StaffID
    FROM Consults
    WHERE PatientID = '2')
```

Expected result:

Name
Tomas
Marta

Aggregation and grouping

Patients by Insurances

```
SELECT Insurances.Name, COUNT(Patients.PatientID)
FROM Patients, Insurances
WHERE Patients.InsuranceID = Insurances.InsuranceID
GROUP BY Insurances.Name
```

Expected result:

Insurance Name	Count
LifeLonger	3
HealthNow	4

Staff by Speciality

```
SELECT Specialities.Title, COUNT(Staff.StaffID)
FROM Specialities, Staff
WHERE Staff.SpecialityID = Specialities.SpecialityID
GROUP BY Specialities.Title
```

Expected result:

Title	Count
Nurse	3
Family Doctor	2

Total cost of consults for specific patient

```
SELECT Patients.Name, SUM (Specialities.Cost)
FROM Consults, Patients, Staff, Specialities
WHERE Consults.PatientID = Patients.PatientID
AND Consults.StaffID = Staff.StaffID
AND Staff.SpecialityID = Specialities.SpecialityID
GROUP BY Patients.Name
```

Expected result:

Patient	Cost
Kevin	340
Louise	400

Implementation

The implementation of this database is done with Visual Studio 2017, C# as main language and SQL Server as our database manager.
Our queries made before were modified during the implementation by following the recommendations of the implementation of SQL in Visual Studio. SQL Server logical processing of a query puts *JOIN* before *WHERE* and it is recommended to use *INNER JOIN*s instead of conditionals if the objective is to link two columns. The performance in our case, would not be a difference, because of the small size of the database, but good manners are learned from the beginning. All the queries that use *WHERE* to join two columns were changed into *JOINs*.

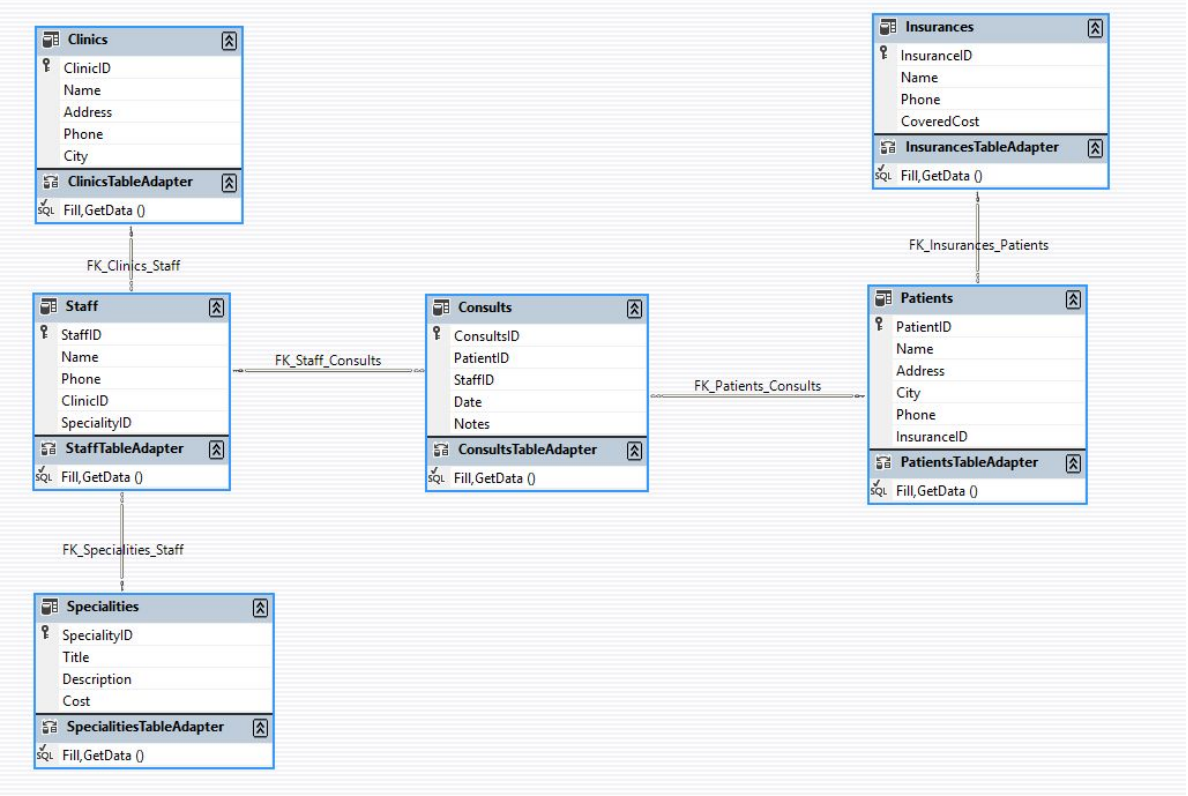
Logical query processing step numbers[2]:

```
(Step 8) SELECT (Step 9) DISTINCT (Step 11)
```

[2]Itzik Ben-Gan. *Inside Microsoft SQL Server 2005: T-SQL Querying (Developer Reference)*, Microsoft Press.

```
(Step 1) FROM left_table
(Step 3) join_type JOIN right_table
(Step 2) ON join_condition
(Step 4) WHERE where_condition
(Step 5) GROUP BY group_by_list
(Step 6) WITH [CUBE|ROLLUP]
(Step 7) HAVING having_clause
(Step 10) ORDER BY order_by_list
```

Foreign Keys
Visual Studio allows the creation of relationships of foreign keys on its data set designer.

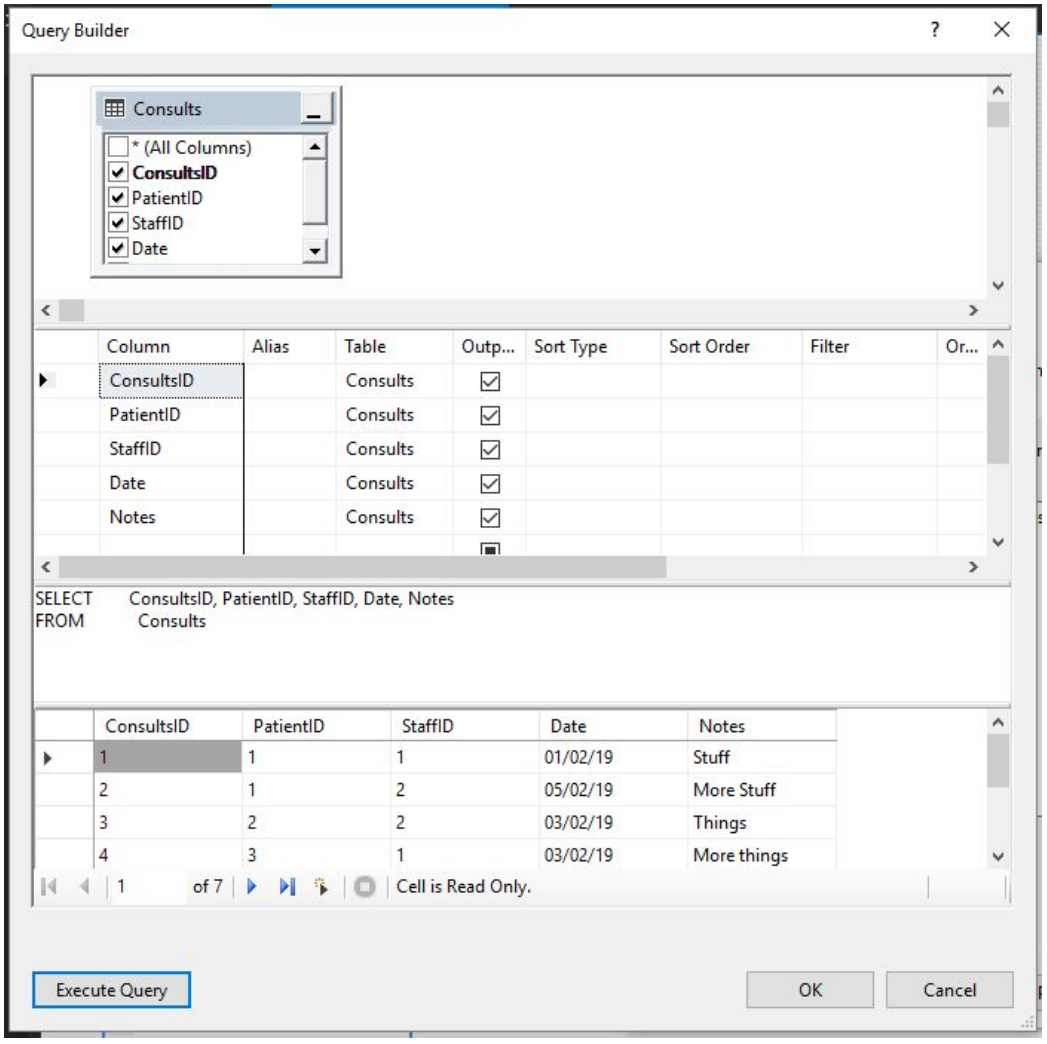


Queries

In order to show how different types of join statements work, query number 5 uses a different join, as explained in its section below.

Raw data
One of the options of the application made is to show raw data, without any special query, for all our tables. This query had no other objective for us that to check the results of other queries.
Visual studio has also deprecated the use of *"*"* in its queries, so in order to show all the columns, they recommend to specify all the columns in the query. For instance, the

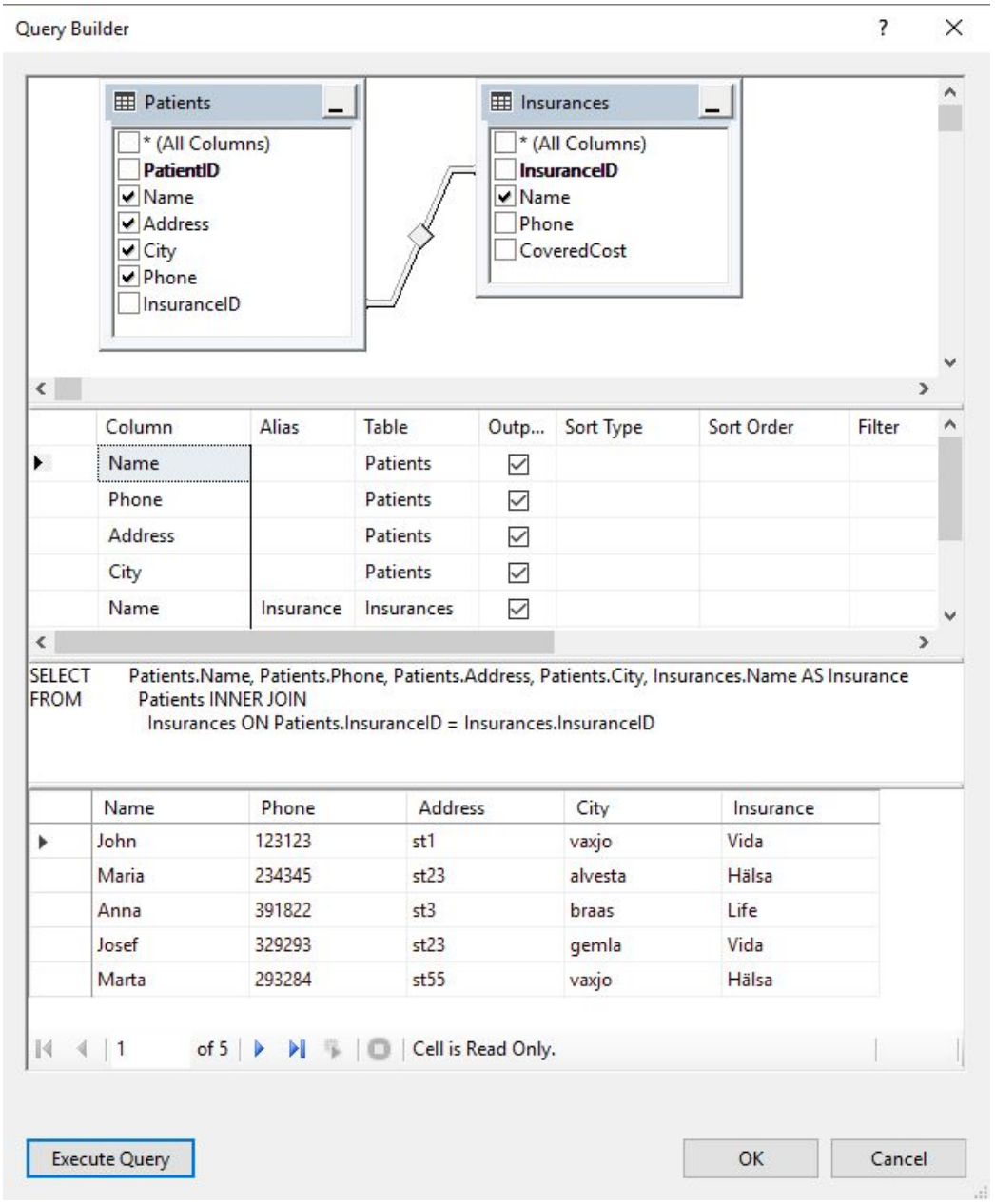
picture below shows the table used to create a query, in this case the table Consults. The query shows all the columns named in *SELECT* for this reason.



All the queries used to show raw data follow this pattern, so we assumed that repeating similar explanations would be redundant in the report.

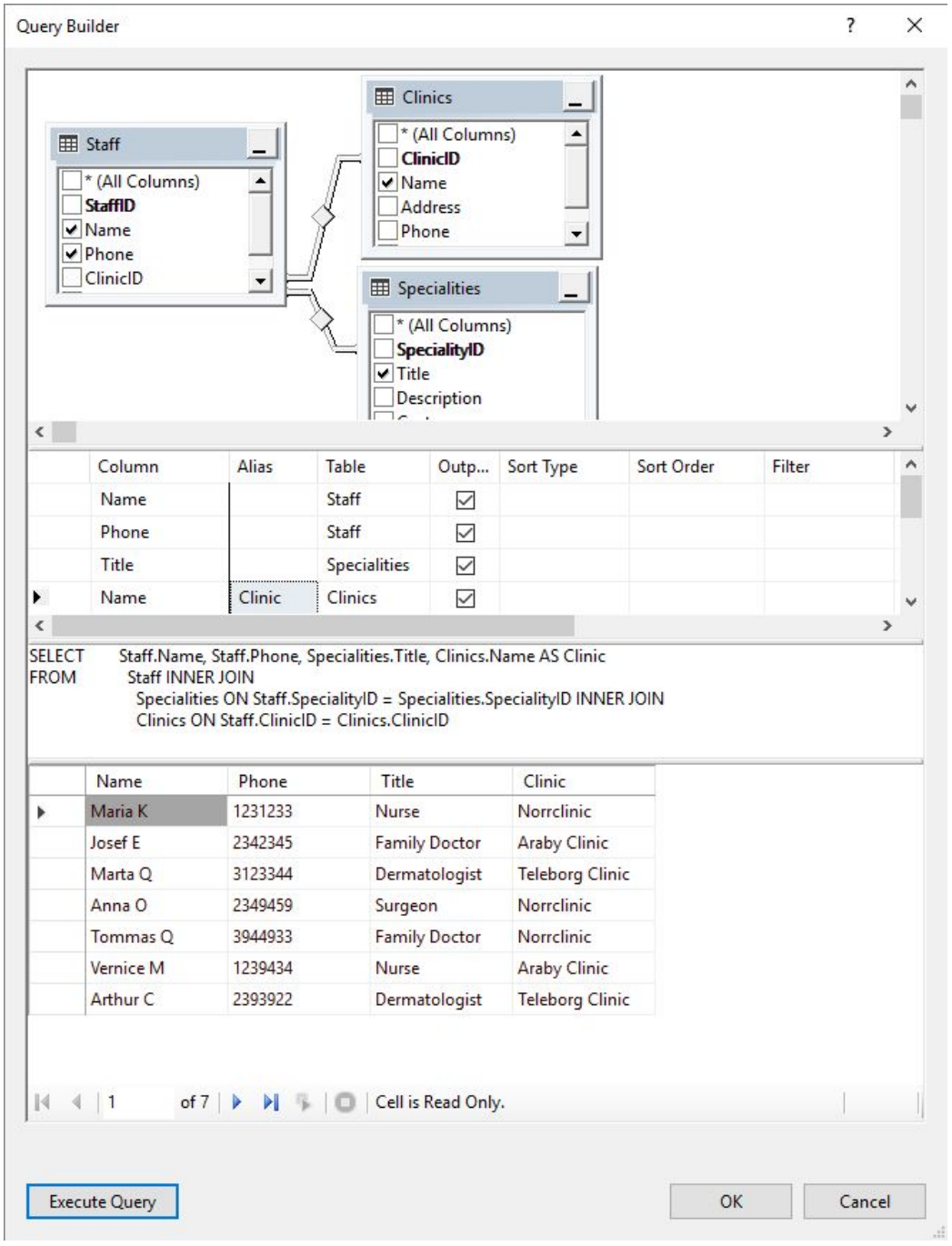
Q1: Patients and Insurances

Show the patients table, but instead of *InsuranceID*, it shows the insurance name.



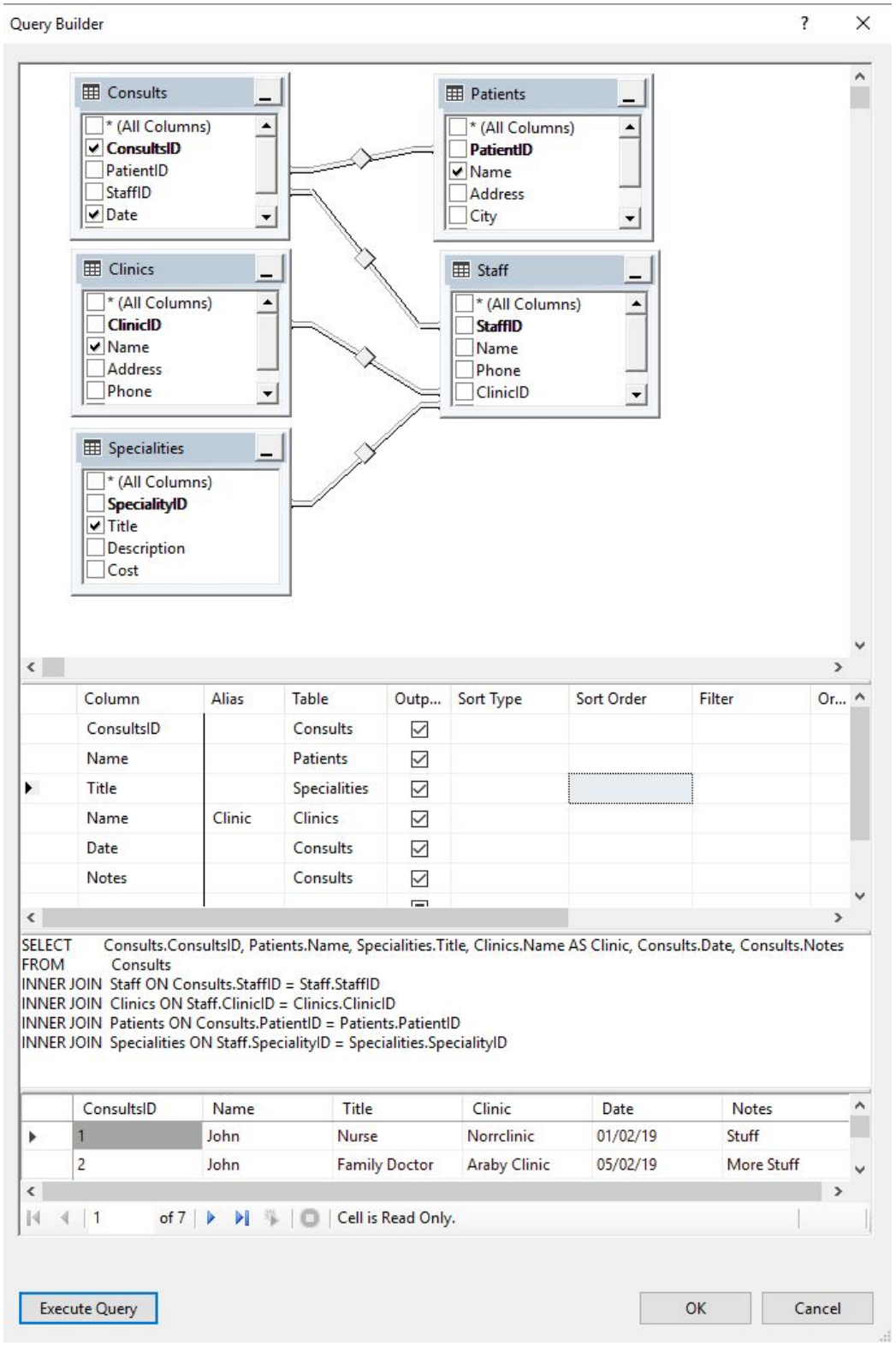
Q2: Staff, specialities and clinics

Show the staff table, but exchange foreign keys with their correspondent name/title attribute.



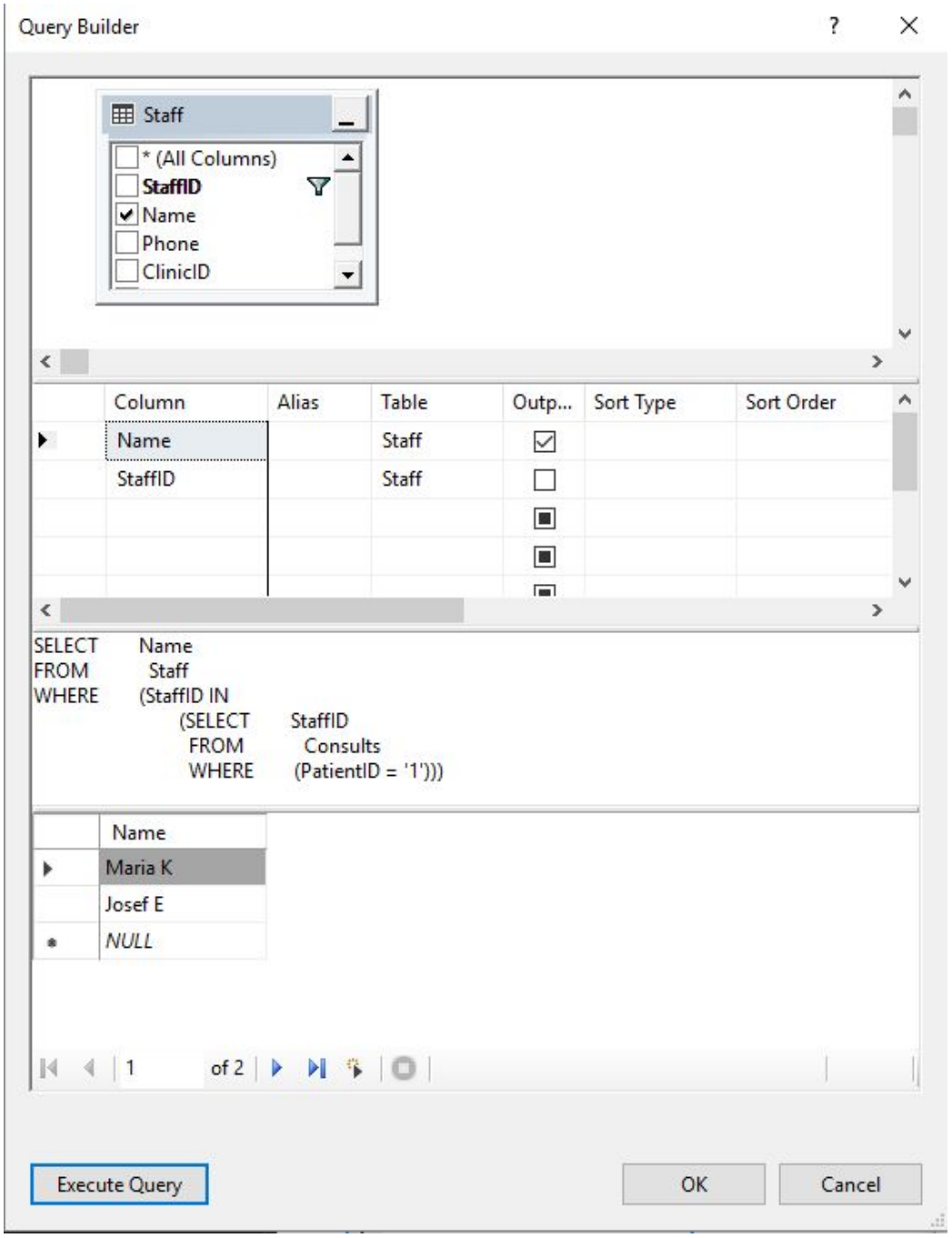
Q3: Consults, the readable table

Show Consults table but exchange *patientID* by their names and *StaffID* by their speciality, adds a column with the name of the staff clinic.



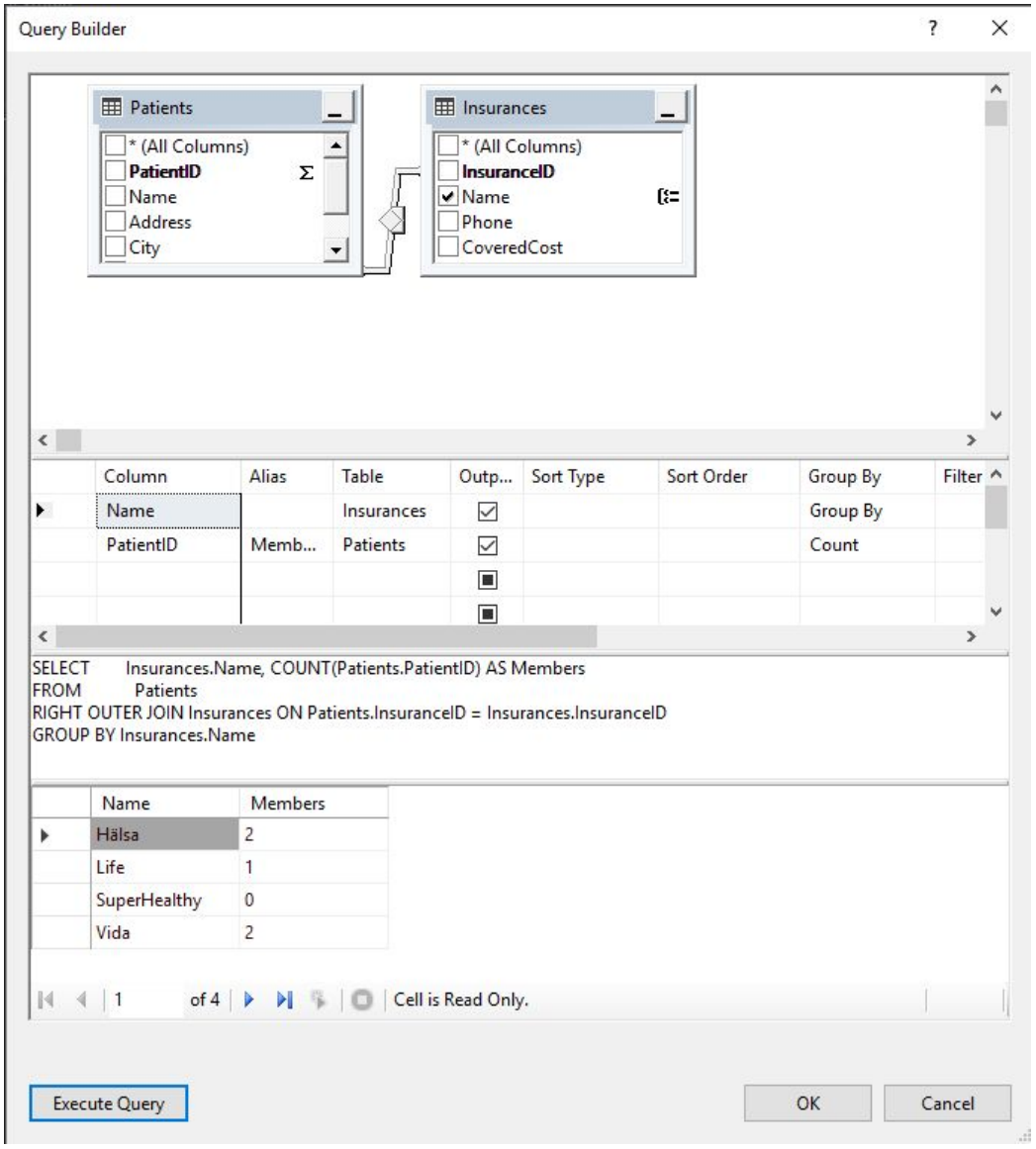
Q4: Staff and a patient

Show the name of all the staff members that has an appointment with an specific patient.



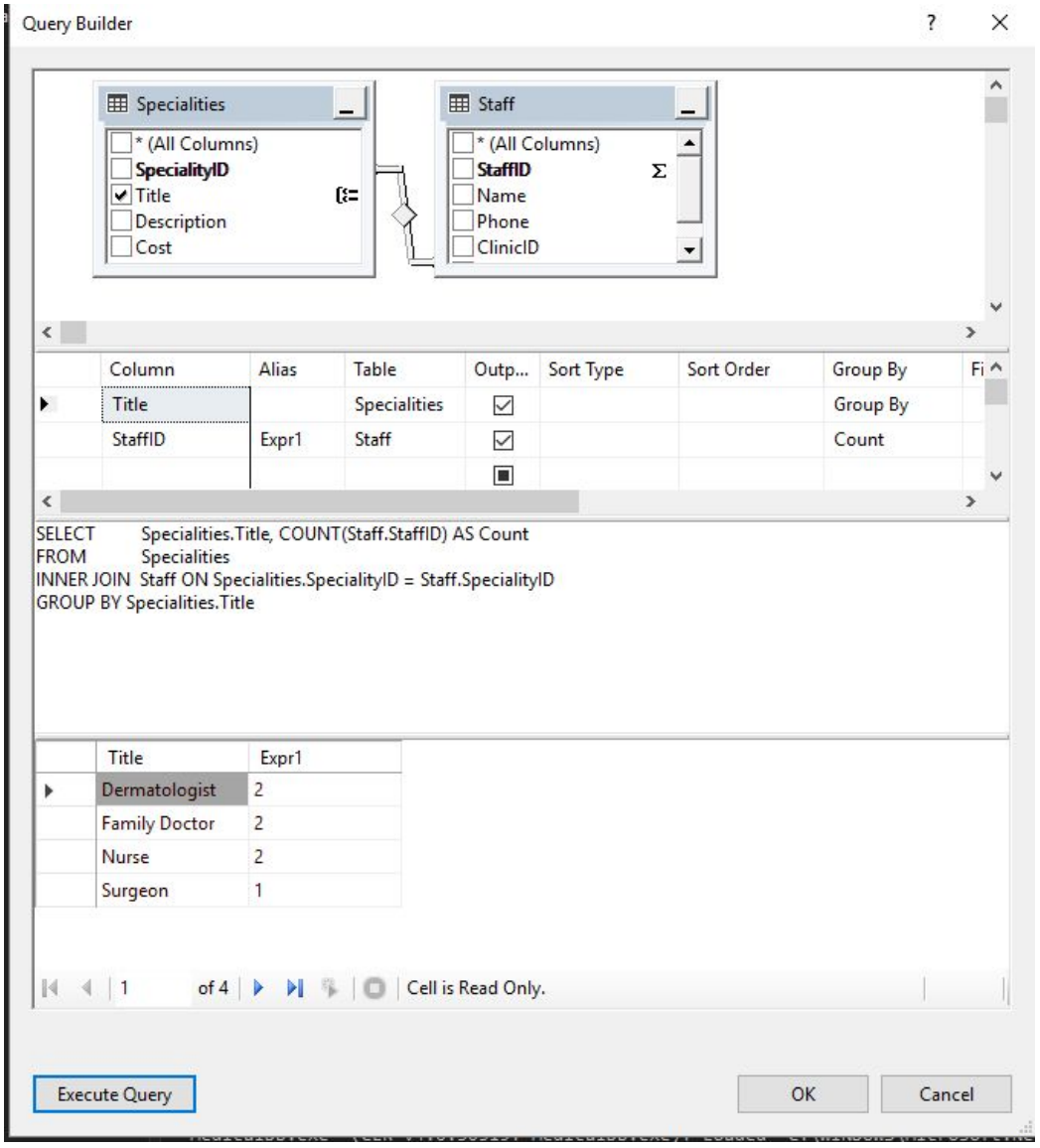
Q5: Patients by Insurances

Instead of following the initial SQL query and in order to show more options in the project, this query was changed to show a different kind of join. Rather than showing a count of patients affiliated to a insurance by a inner join, here we used a right join. The difference is that the right table (Insurances, as shows the picture below) will return all the records and the matched ones from the left table (Patients). The last panel that shows the result table shows the name of insurances even when there is no match members (SuperHealthy Insurance has zero matches, but is still in the result).



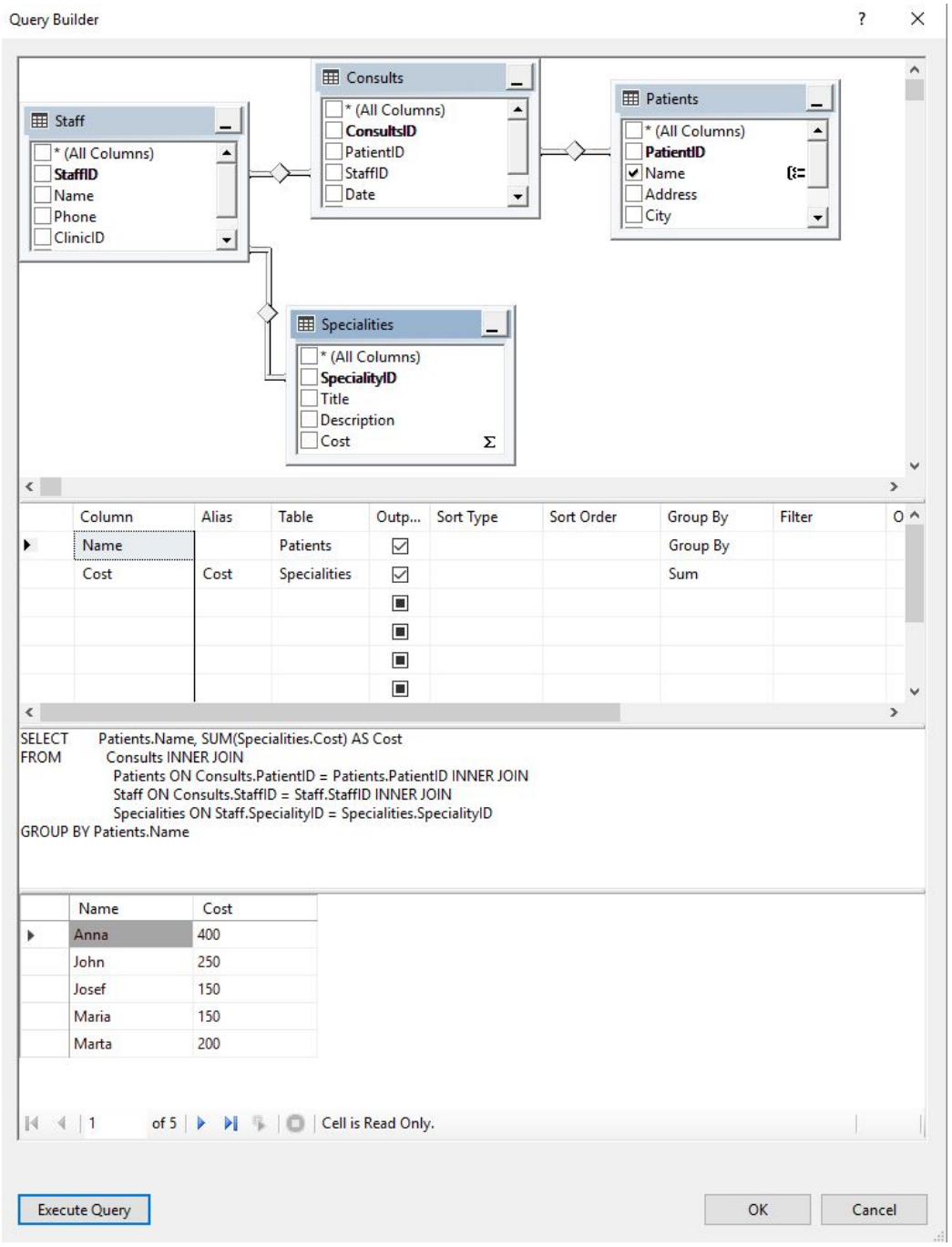
Q6: Staff by Speciality

Show how many members of the staff are by speciality



Q7: Total cost of consults by patient

Shows how much will cost the total of consults to each patient.



Instructions

To run the MedicalDB.exe the following prerequisites must be installed.

Prerequisites:

- .NET Framework 4.6 - 4.6 development tools (and later versions)
- SQL Server 2016 Express LocalDB

When all is installed, run the MedicalDB.exe file or import the src files into a suitable IDE for C# and run the Program.cs file.

Supplemental Video

Watch the whole video for short explanations, or if interested only into running the app, start at the minute 01:53

https://youtu.be/xyYbrVAz_Co