



Report

Database Theory

Final Project



Författare: Ruth Dirnfeld

Helena Tevar

Handledare: Maria Ullan

Examinator: Illir Jusufi

Termin: HT18

Ämne: Database Theory

Kurskod: 2DV513



Innehållsförteckning

Innehållsförteckning	2
Idea	3
Logical Model	3
Design in SQL	4
Tables creation	4
Patients	4
Insurances	4
Staff	5
Specialities	5
Clinics	5
Consults	5
SQL Queries	6
Patients and Insurances	6
Staff, specialities and clinics	6
Consults, the readable table	6
Staff and a patient	7
Aggregation and grouping	7
Patients by Insurances	7
Staff by Speciality	8
Total cost of consults for specific patient	8
Implementation	8
Foreign Keys	9
Queries	9
Raw data	9
Q1: Patients and Insurances	11
Q2: Staff, specialities and clinics	12
Q3: Consults, the readable table	13
Q4: Staff and a patient	14
Q5: Patients by Insurances	15
Q6: Staff by Speciality	16
Q7: Total cost of consults by patient	17
Instructions	17
Supplemental Video	18



Idea

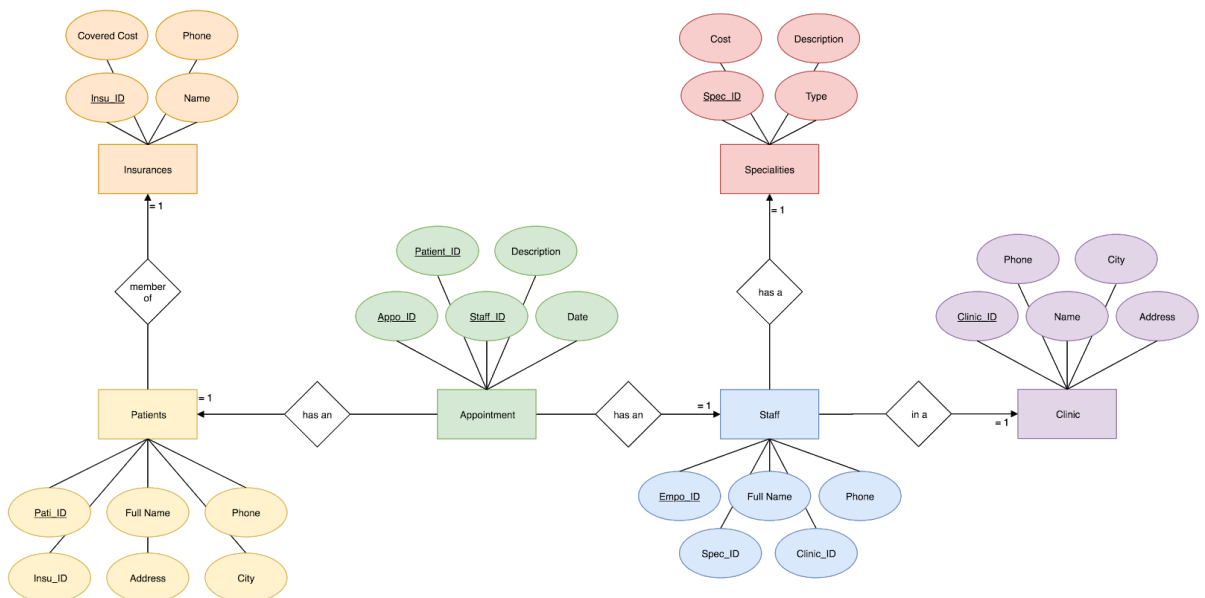
The project idea was to create a database of a medical clinic or group of clinic to be used by their staff. For instance, administrative staff, to manage patients, medical staff and the appointments.

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 explanatory.

```
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, specialties 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 JOINS* 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 numbers²:

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

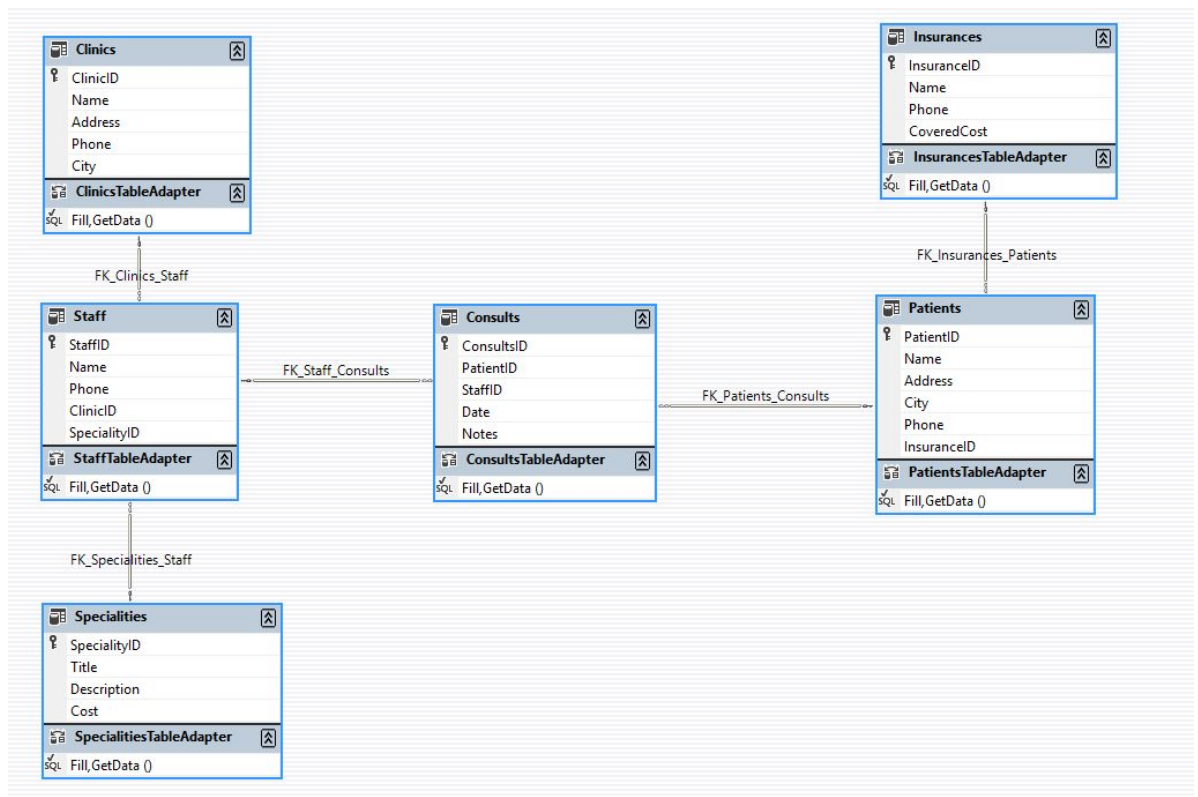
²Itzik Ben-Gan. *Inside Microsoft SQL Server 2005: T-SQL Querying (Developer Reference)*, Microsoft Press.



```
(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.

The screenshot shows the 'Query Builder' window. At the top, the 'Consults' table is selected. Below it, a list of columns is shown with checkboxes: ☐ * (All Columns), ☒ ConsultsID, ☒ PatientID, ☒ StaffID, and ☒ Date. The main area displays a table with columns: Column, Alias, Table, Outp..., Sort Type, Sort Order, Filter, and Or... The data rows are:

Column	Alias	Table	Outp...	Sort Type	Sort Order	Filter	Or...
ConsultsID		Consults	<input checked="" type="checkbox"/>				
PatientID		Consults	<input checked="" type="checkbox"/>				
StaffID		Consults	<input checked="" type="checkbox"/>				
Date		Consults	<input checked="" type="checkbox"/>				
Notes		Consults	<input checked="" type="checkbox"/>				

Below the table, the SQL query is displayed:

```
SELECT ConsultsID, PatientID, StaffID, Date, Notes
FROM Consults
```

At the bottom, a preview table shows the results of the query:

ConsultsID	PatientID	StaffID	Date	Notes
1	1	1	01/02/19	Stuff
2	1	2	05/02/19	More Stuff
3	2	2	03/02/19	Things
4	3	1	03/02/19	More things

The bottom of the window includes an 'Execute Query' button, 'OK' and 'Cancel' buttons, and a status bar showing '1 of 7' and 'Cell is Read Only.'

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.

Query Builder

Patients

- ☐ * (All Columns)
- ☐ PatientID
- ☒ Name
- ☒ Address
- ☒ City
- ☒ Phone
- ☐ InsuranceID

Insurances

- ☐ * (All Columns)
- ☐ InsuranceID
- ☒ Name
- ☐ Phone
- ☐ CoveredCost

	Column	Alias	Table	Outp...	Sort Type	Sort Order	Filter
▶	Name		Patients	<input checked="" type="checkbox"/>			
	Phone		Patients	<input checked="" type="checkbox"/>			
	Address		Patients	<input checked="" type="checkbox"/>			
	City		Patients	<input checked="" type="checkbox"/>			
	Name	Insurance	Insurances	<input checked="" type="checkbox"/>			

SELECT Patients.Name, Patients.Phone, Patients.Address, Patients.City, Insurances.Name AS Insurance
FROM Patients INNER JOIN
Insurances ON Patients.InsuranceID = Insurances.InsuranceID

	Name	Phone	Address	City	Insurance
▶	John	123123	st1	vaxjo	Vida
	Maria	234345	st23	alvesta	Hälsa
	Anna	391822	st3	braas	Life
	Josef	329293	st23	gemla	Vida
	Marta	293284	st55	vaxjo	Hälsa

1 of 5 | Cell is Read Only.

Execute Query OK Cancel



Q2: Staff, specialties and clinics

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

Query Builder

The Query Builder interface shows three tables: Staff, Specialties, and Clinics. The Staff table is selected, and its columns (Name, Phone, ClinicID) are joined to the Specialties table (Title, Description) and the Clinics table (Name, Address, Phone). The query is executed, and the results are displayed in a table.

Column	Alias	Table	Outp...	Sort Type	Sort Order	Filter
Name		Staff	<input checked="" type="checkbox"/>			
Phone		Staff	<input checked="" type="checkbox"/>			
Title		Specialties	<input checked="" type="checkbox"/>			
Name	Clinic	Clinics	<input checked="" type="checkbox"/>			

SELECT Staff.Name, Staff.Phone, Specialties.Title, Clinics.Name AS Clinic
FROM Staff INNER JOIN
Specialties ON Staff.SpecialtyID = Specialties.SpecialtyID INNER JOIN
Clinics ON Staff.ClinicID = Clinics.ClinicID

Name	Phone	Title	Clinic
Maria K	1231233	Nurse	Norrclinic
Josef E	2342345	Family Doctor	Araby Clinic
Marta Q	3123344	Dermatologist	Teleborg Clinic
Anna O	2349459	Surgeon	Norrclinic
Tommas Q	3944933	Family Doctor	Norrclinic
Vernice M	1239434	Nurse	Araby Clinic
Arthur C	2393922	Dermatologist	Teleborg Clinic

1 of 7 | Cell is Read Only.

Execute Query OK Cancel



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.

Query Builder

The diagram shows the following relationships:

- Consults (PK: ConsultsID) is connected to Patients (FK: PatientID).
- Consults (FK: StaffID) is connected to Staff (FK: StaffID).
- Consults (FK: ClinicID) is connected to Clinics (FK: ClinicID).
- Consults (FK: SpecialityID) is connected to Specialities (FK: SpecialityID).

Selected columns in the diagram:

- Consults: ConsultsID, Date
- Patients: Name
- Clinics: Name
- Specialities: Title
- Staff: StaffID, ClinicID

Column	Alias	Table	Outp...	Sort Type	Sort Order	Filter	Or...
ConsultsID		Consults	<input checked="" type="checkbox"/>				
Name		Patients	<input checked="" type="checkbox"/>				
Title		Specialities	<input checked="" type="checkbox"/>				
Name	Clinic	Clinics	<input checked="" type="checkbox"/>				
Date		Consults	<input checked="" type="checkbox"/>				
Notes		Consults	<input checked="" type="checkbox"/>				

```
SELECT Consults.ConsultsID, Patients.Name, Specialities.Title, Clinics.Name AS Clinic, Consults.Date, Consults.Notes
FROM Consults
INNER JOIN Staff ON Consults.StaffID = Staff.StaffID
INNER JOIN Clinics ON Staff.ClinicID = Clinics.ClinicID
INNER JOIN Patients ON Consults.PatientID = Patients.PatientID
INNER JOIN Specialities ON Staff.SpecialityID = Specialities.SpecialityID
```

ConsultsID	Name	Title	Clinic	Date	Notes
1	John	Nurse	Norrclinic	01/02/19	Stuff
2	John	Family Doctor	Araby Clinic	05/02/19	More Stuff

1 of 7 | Cell is Read Only.

Execute Query OK Cancel



Q4: Staff and a patient

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

Query Builder

Staff

- ☐ * (All Columns)
- ☐ StaffID
- ☒ Name
- ☐ Phone
- ☐ ClinicID

	Column	Alias	Table	Outp...	Sort Type	Sort Order
▶	Name		Staff	<input checked="" type="checkbox"/>		
	StaffID		Staff	<input type="checkbox"/>		
				<input type="checkbox"/>		
				<input type="checkbox"/>		
				<input type="checkbox"/>		

```
SELECT Name
FROM Staff
WHERE (StaffID IN
      (SELECT StaffID
       FROM Consults
       WHERE (PatientID = '1')))
```

	Name
▶	Maria K
	Josef E
*	NULL

1 of 2

Execute Query OK Cancel



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 an 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).

Query Builder

Column	Alias	Table	Outp...	Sort Type	Sort Order	Group By	Filter
Name		Insurances	<input checked="" type="checkbox"/>			Group By	
PatientID	Memb...	Patients	<input checked="" type="checkbox"/>			Count	

```
SELECT  Insurances.Name, COUNT(Patients.PatientID) AS Members
FROM    Patients
RIGHT OUTER JOIN Insurances ON Patients.InsuranceID = Insurances.InsuranceID
GROUP BY Insurances.Name
```

Name	Members
Hälsa	2
Life	1
SuperHealthy	0
Vida	2

1 of 4 | Cell is Read Only.

Execute Query OK Cancel



Q6: Staff by Speciality

Show how many members of the staff are by speciality

Query Builder

Specialities

- ☐ * (All Columns)
- ☐ SpecialityID
- ☒ Title
- ☐ Description
- ☐ Cost

Staff

- ☐ * (All Columns)
- ☐ StaffID
- ☐ Name
- ☐ Phone
- ☐ ClinicID

Column Alias Table Outp... Sort Type Sort Order Group By

Title		Specialities	<input checked="" type="checkbox"/>			Group By
StaffID	Expr1	Staff	<input checked="" type="checkbox"/>			Count

SELECT Specialities.Title, COUNT(Staff.StaffID) AS Count
FROM Specialities
INNER JOIN Staff ON Specialities.SpecialityID = Staff.SpecialityID
GROUP BY Specialities.Title

Title	Expr1
Dermatologist	2
Family Doctor	2
Nurse	2
Surgeon	1

1 of 4 Cell is Read Only.

Execute Query OK Cancel



Q7: Total cost of consults by patient

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

Query Builder

Staff: * (All Columns), StaffID, Name, Phone, ClinicID

Consults: * (All Columns), ConsultsID, PatientID, StaffID, Date

Patients: * (All Columns), PatientID, Name, Address, City

Specialities: * (All Columns), SpecialityID, Title, Description, Cost

Column	Alias	Table	Outp...	Sort Type	Sort Order	Group By	Filter
Name		Patients	<input checked="" type="checkbox"/>			Group By	
Cost	Cost	Specialities	<input checked="" type="checkbox"/>			Sum	

```
SELECT Patients.Name, SUM(Specialities.Cost) AS Cost
FROM Consults INNER JOIN
    Patients ON Consults.PatientID = Patients.PatientID INNER JOIN
    Staff ON Consults.StaffID = Staff.StaffID INNER JOIN
    Specialities ON Staff.SpecialityID = Specialities.SpecialityID
GROUP BY Patients.Name
```

Name	Cost
Anna	400
John	250
Josef	150
Maria	150
Marta	200

1 of 5 | Cell is Read Only.

Execute Query OK Cancel



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