

Data normalization

The normalization process aims to minimize data duplications, avoid errors during data modifications, and simplify data queries from the database. The three fundamental normalization forms are known as:

First Normal Form (1NF)

Second Normal Form (2NF)

Third Normal Form (3NF)

In this reading, you will learn how to apply the rules that ensure that a database meets the criteria of these three normal forms.

The following example includes fictitious data required by a Medical Group Surgery based in London to generate relevant reports. Doctors work in multiple regions and various councils in London. And once patients book an appointment, they are given a slot ID at their local surgery. There might be multiple surgeries in the same council but with different postcodes, where one or more councils belong to a particular region. For example, East or West London.

Doctor ID	Doctor name	Region	Patient ID	Patient name	Surgery Number	Surgery council	Postcode	Slot ID	Total Cost
D1	Karl	West London	P1	Rami	3	Harrow	HA9SDEA	A1	1500 1200
			P2	Kim				A2	1600
			P3	Nora				A3	
D1	Karl	East London	P4	Kamel	4	Hackney	E1 6AW	A1	2500 1000
			P5	Sami				A2	
D2	Mark	East London	P5	Sami	4	Hackney	E1 6AW	A3	1500 2000
			P6	Norma				A4	
D2	Mark	West London	P7	Rose	5	Harrow	HA862E	A4	1000
			P1	Rami				A5	1500

The data listed in the table are in an unnormalized form. Repeating groups of data appear in many cases, for instance, doctors, regions, and council names. There are also multiple instances of data stored in the same cell, such as with the patient name and total cost columns. This makes it difficult to update and query data. Moreover, it is not easy to choose a unique key and assign it as a primary key.

This unnormalized table can be written in SQL form as follows:

```
1 CREATE TABLE Surgery
2   (DoctorID VARCHAR(10),
3    DoctorName VARCHAR(50),
4    Region VARCHAR(20),
5    PatientID VARCHAR(10),
6    PatientName VARCHAR(50),
7    SurgeryNumber INT, Council VARCHAR(20),
8    Postcode VARCHAR(10),
9    SlotID VARCHAR(5),
10   TotalCost Decimal);
```

First normal form

To simplify the data structure of the surgery table, let's apply the first normal form rules to enforce the data atomicity rule and eliminate unnecessary repeating data groups. The data atomicity rule means you can only have one single instance value of the column attribute in any table cell.

The atomicity problem only exists in the columns of data related to the patients. Therefore, it is important to create a new table for patient data to fix this. In other words, you can organize all data related to the patient entity in one separate table, where each column cell contains only one single instance of data, as depicted in the following example:

Patient ID	Patient name	Slot ID	Total Cost
P1	Rami	A1	1500

Patient ID	Patient name	Slot ID	Total Cost
P2	Kim	A2	1200
P3	Nora	A3	1600
P4	Kamel	A1	2500
P5	Sami	A2	1000
P6	Norma	A5	2000
P7	Rose	A6	1000

This table includes one single instance of data in each cell, which makes it much simpler to read and understand. However, the patient table requires two columns, the patient ID and the Slot ID, to identify each record uniquely. This means that you need a composite primary key in this table. To create this table in SQL you can write the following code:

```

1 CREATE TABLE Patient
2   (PatientID VARCHAR(10) NOT NULL,
3    PatientName VARCHAR(50),
4    SlotID VARCHAR(10) NOT NULL,
5    TotalCost Decimal,
6    CONSTRAINT PK_Patient
7    PRIMARY KEY (PatientID, SlotID));

```

Once you have removed the patient attributes from the main table, you just have the doctor ID, name, region, surgery number, council and postcode columns left in the table.

Doctor ID	Doctor name	Region	Surgery Number	Surgery council	Postcode
D1	Karl	West London	3	Harrow	HA9SDE
D1	Karl	East London	4	Hackney	E1 6AW
D2	Mark	West London	4	Hackney	E1 6AW
D2	Mark	East London	5	Harrow	HA862E

You may have noticed that the table also contains repeating groups of data in each column. You can fix this by separating the table into two tables of data: the doctor table and the surgery table, where each table deals with one specific entity.

Doctor table

Doctor ID	Doctor name
D1	Karl
D2	Mark

Surgery table

Surgery Number	Region	Surgery council	Postcode
3	West London	Harrow	HA9SDE
4	East London	Hackney	E1 6AW
5	West London	Harrow	HA862E

In the doctor table, you can identify the doctor ID as a single-column primary key. This table can be created in SQL by writing the following code:

```

1 CREATE TABLE Doctor
2   (DoctorID VARCHAR(10),
3    DoctorName VARCHAR(50), PRIMARY KEY (DoctorID)
4   );

```

Similarly, the surgery table can have the surgery number as a single-column primary key. The surgery table can be created in SQL by writing the following code:

```

1 CREATE TABLE Surgery
2 (SurgeryNumber INT NOT NULL,
3 Region VARCHAR(20), Council VARCHAR(20),
4 Postcode VARCHAR(10), PRIMARY KEY (SurgeryNumber)
5 );

```

By applying the atomicity rule and removing the repeating data groups, the database now meets the first normal form.

Second normal form

In the second normal form, you must avoid partial dependency relationships between data. Partial dependency refers to tables with a composite primary key. Namely, a key that consists of a combination of two or more columns, where a non-key attribute value depends only on one part of the composite key.

Since the patient table is the only one that includes a composite primary key, you only need to look at the following table.

<u>Patient ID</u>	<u>Patient name</u>	<u>Slot ID</u>	<u>Total Cost</u>
P1	Rami	A1	1500
P2	Kim	A2	1200
P3	Nora	A3	1600
P4	Kamel	A1	2500
P5	Sami	A2	1000
P5	Sami	A3	1000
P6	Sami	A4	1500
P7	Norma	A5	2000
P8	Rose	A6	1000
P1	Rami	A7	1500

In the patient table, you must check whether any non-key attributes depend on one part of the composite key. For example, the patient's name is a non-key attribute, and it can be determined by using the patient ID only.

Similarly, you can determine the total cost by using the Slot ID only. This is called partial dependency, which is not allowed in the second normal form. This is because all non-key attributes should be determined by using both parts of the composite key, not only one of them.

This can be fixed by splitting the patient table into two tables: patient table and appointment table. In the patient table you can keep the patient ID and the patient's name.

Patient table

<u>Patient ID</u>	<u>Patient name</u>
P1	Rami
P2	Kim
P3	Nora
P4	Kamel
P5	Sami
P7	Norma
P8	Rose

The new patient table can be created in SQL using the following code:

```

1 CREATE TABLE Patient
2 (PatientID VARCHAR(10) NOT NULL,
3 PatientName, VARCHAR(50), PRIMARY KEY (PatientID)
4 );

```

However, in the appointment table, you need to add a unique key to ensure you have a primary key that can identify each unique record in the table. Therefore, the appointment ID attribute can be added to the table with a unique value in each row.

Appointment table before adding a new unique key		Appointment table after adding a new unique key		
Appointment table		Appointment table		
Slot ID	Total Cost	Appointment ID	Slot ID	Total Cost
A1	1500	1	A1	1500
A2	1200	2	A2	1200
A3	1600	3	A3	1600
A1	2500	4	A1	2500
A2	1000	5	A2	1000
A3	1000	6	A3	1000
A4	1500	7	A4	1500
A5	2000	8	A5	2000
A6	1000	9	A6	1000
A7	1500	10	A7	1500

The new appointments table can be created in SQL using the following code:

```

1 CREATE TABLE Appointments
2 (AppointmentID INT NOT NULL,
3 SlotID, VARCHAR(10),
4 TotalCost Decimal, PRIMARY KEY (AppointmentID)
5 );

```

You have removed the partial dependency, and all tables conform to the first and second normal forms.

Third normal form

For a relation in a database to be in the third normal form, it must already be in the second normal form (2NF). In addition, it must have no transitive dependency. This means that any non-key attribute in the surgery table may not be functionally dependent on another non-key attribute in the same table. In the surgery table, the postcode and the council are non-key attributes, and the postcode depends on the council. Therefore, if you change the council value, you must also change the postcode. This is called transitive dependency, which is not allowed in the third normal form.

Surgery number	Region	Surgery council	Postcode
3	West London	Harrow	HA9SDE
4	East London	Hackney	E1 6AW
5	West London	Harrow	HA862E

In other words, changing the value of the council value in the above table has a direct impact on the postcode value, because each postcode in this example belongs to a specific council. This transitive dependency is not allowed in the third normal form. To fix it you can split this table into two tables: one for the region with the city and one for the surgery.

Location table

Surgery number	Postcode
3	HA9SDE
4	E1 6AW
5	HA862E

The new surgery location table can be created in SQL using the following code:

```

1 CREATE TABLE Location
2 (SurgeryNumber INT NOT NULL,
3  Postcode VARCHAR(10), PRIMARY KEY (SurgeryNumber)
4 );

```

Council table Surgery council

Harrow

Hackney

Region

West London

East London

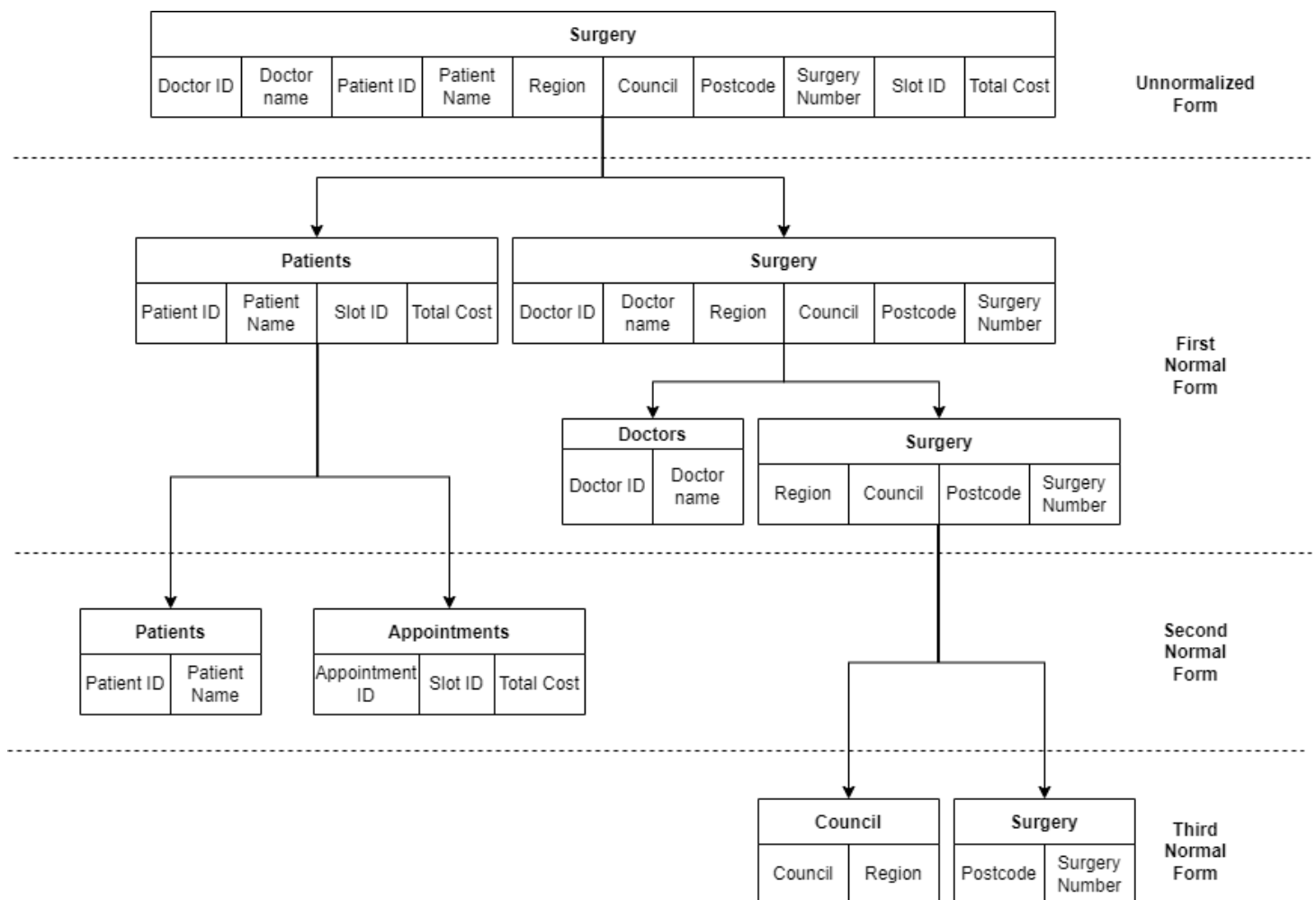
The new surgery council table can be created in SQL using the following code:

```

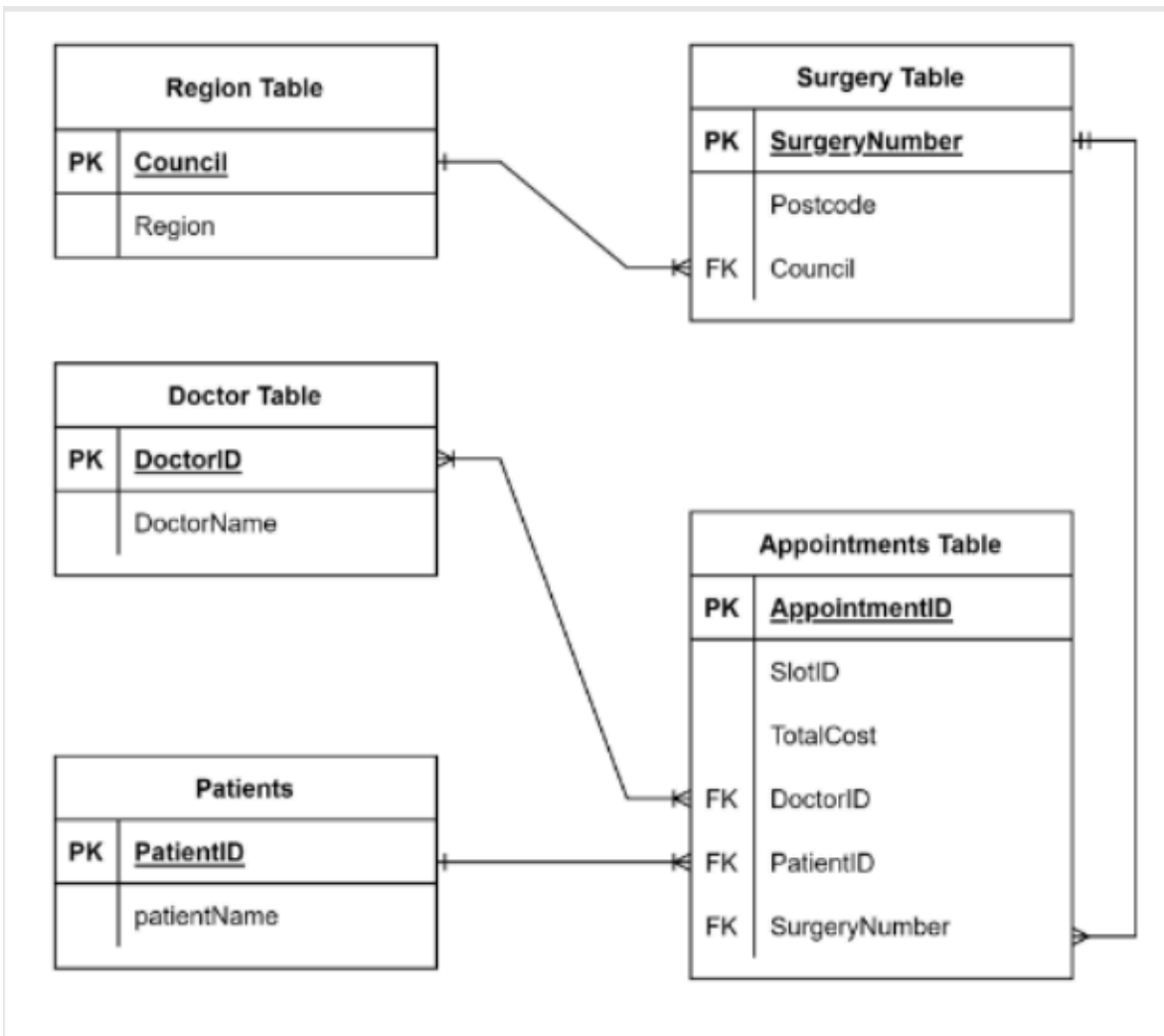
1 CREATE TABLE Council
2 (Council VARCHAR(20) NOT NULL,
3  Region VARCHAR(20), PRIMARY KEY (Council)
4 );

```

This ensures the database conforms to first, second, and third normal forms. The following diagram illustrates the stages through which the data moves from the unnormalized form to the first normal form, the second normal form, and finally to the third normal form.



However, it's important to link all tables together to ensure you have well-organized and related tables in the database. This can be done by defining foreign keys in the tables.



The third normal form is typically good enough to deal with the three anomaly challenges – insertion, update, and deletion anomalies – that the normalization process aims to tackle. Completing the third normal form in a database design helps to develop a database that is easy to access and query, well-structured, well-organized, consistent, and without unnecessary data duplications.

Mark as completed