A Complete Guide to Database Normalization in SQL

Imagine you've just been asked to manage your company's relational database system. Eager to impress, you quickly run a few initial queries to familiarize yourself with the data… only to find the tables in organizational disarray.

You freeze. You're worried about the negative impact the inconsistent dependencies could have on future data manipulation queries and long-term analyses. But you're also unsure of what steps to take to correctly redesign the tables. And suddenly the unwelcome urge to dig through your notes from the database management course you took a lifetime ago begins to plague you.

Sound familiar?

Don't panic. Whether you have inherited a messy database, unintentionally synthesized one with poor integrity (whoops! 😬), or want to avoid the above scenario altogether, **database normalization** is the solution for you.

What is Database Normalization?

According to the database normalization page on Wikipedia:

"Normalization entails organizing the columns (attributes) and tables (relations) of a database to ensure that their

dependencies are properly enforced by database integrity constraints.”

Yikes.

Don’t let these types of definitions scare you off. Translated into plain English, this simply means that **normalization is the process of creating a maximally efficient relational database.** Essentially, databases should be organized to decrease redundancy and avoid dependence anomalies.

In even simpler terms, *your database structure should make intuitive sense*. If a fellow coworker is terrified of making a fatal error while working with a database you’ve created even after you explain it to them, your database probably isn’t normalized.

These normalization ideals can be applied to database **synthesis** (creating a database from scratch) or **decomposition** (improving existing designs).

What are Normal Forms?

“Normalization” is a broad concept and isn’t much practical use when you’re lost at sea among a myriad of messy tables.

A Complete Guide to Database Normalization in SQL

Codd's normalization guidelines have five official normal forms, but (thankfully) the first three are usually as in-depth as you need to go. Let's briefly review these here:

First Normal Form (1NF)

This initial set of rules sets the fundamental guidelines for keeping your database properly organized.

- Remove any repeating groups of data *(i.e. beware of duplicative columns or rows within the same table)*
- Create separate tables for each group of related data
- Each table should have a primary key (*i.e. a field that identifies each row with a non-null, unique value*)

Second Normal Form (2NF)

This next set of rules builds upon those outlined in 1NF.

- Meet every rule from 1NF
- Remove data that doesn't depend on the table's primary key *(either move the data to the appropriate table or create a new table and primary key)*
- Foreign keys are used to identify table relationships

A Complete Guide to Database Normalization in SQL

Third Normal Form (3NF)

This set of rules takes those outlined in 1NF and 2NF a step further.

- Meet every rule from 1NF and 2NF
- Remove attributes that rely on other non-key attributes *(i.e. remove columns that depend on columns that aren't foreign or primary keys)*

Why Should You Care?

Yes, high-end database normalization is often considered a luxury and not an absolute requirement. But even small steps towards the right direction can help avoid slow degradation of data integrity over time.

Ensuring your database dependencies make logical sense and redundancy is minimized likewise ensures maximally insightful queries and analysis.

Normalization also combats data manipulation (think DELETE, INSERT, and UPDATE) anomalies. If dependencies aren't normalized, you run the risk of allowing for partially updated (and, therefore, partially incorrect) data. Partially incorrect data = partially incorrect query results down the line.

A Complete Guide to Database Normalization in SQL

A Database Normalization Example

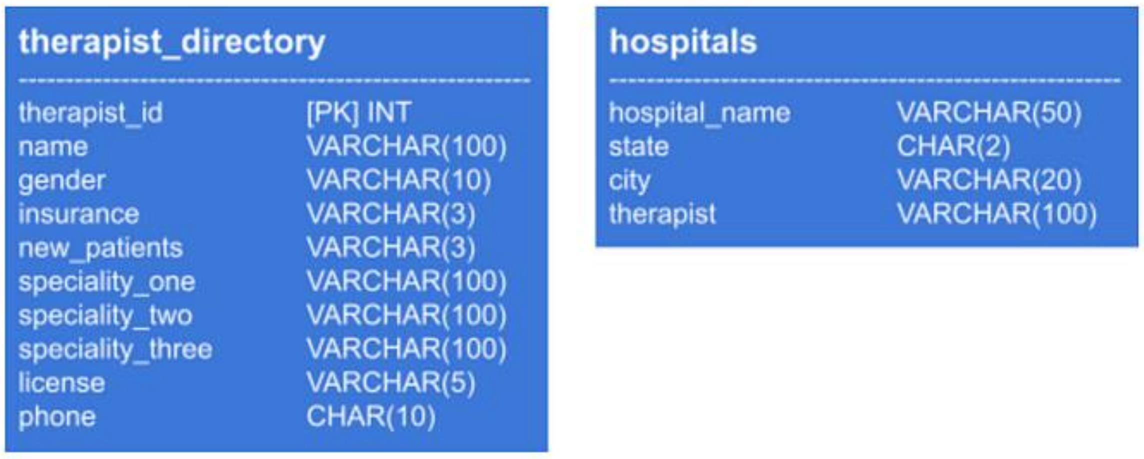
To fully let these abstract definitions sink in, let's review each normal form with a concrete example. We'll be focusing on *decomposition* throughout the examples, but the concepts still apply to synthesis-based projects as well.

For these examples, I personally utilized MySQL But, again, if you prefer an alternative SQL server such as Oracle, Snowflake you can translate the techniques reviewed here to your platform of choice.

The Data

Let's pretend you have been hired at a company that has a database with information regarding therapists located in California. For the purpose of this tutorial, I have created a mock SQL database hosted through MySQL with data that attempts to emulate a very small slice of what a similar, real database might contain. The database has the following tables:

A Complete Guide to Database Normalization in SQL



therapist_directory	
therapist_id	[PK] INT
name	VARCHAR(100)
gender	VARCHAR(10)
insurance	VARCHAR(3)
new_patients	VARCHAR(3)
speciality_one	VARCHAR(100)
speciality_two	VARCHAR(100)
speciality_three	VARCHAR(100)
license	VARCHAR(5)
phone	CHAR(10)

hospitals	
hospital_name	VARCHAR(50)
state	CHAR(2)
city	VARCHAR(20)
therapist	VARCHAR(100)

Original database schema

```
CREATE TABLE IF NOT EXISTS therapist_directory(
	therapist_id INT PRIMARY KEY,
	name VARCHAR(100),
	gender VARCHAR(10),
	insurance VARCHAR(3) CHECK(insurance IN ('Yes', 'No')),
	new_patients VARCHAR(3) CHECK(insurance IN ('Yes', 'No')),
	speciality_one VARCHAR(100),
	speciality_two VARCHAR(100),
	speciality_three VARCHAR(100),
	license VARCHAR(5) CHECK(license IN ('MFT', 'PhD', 'MD')),
	phone CHAR(10)
);
```

```
CREATE TABLE IF NOT EXISTS hospitals(
	hospital_name VARCHAR(50),
	state CHAR(2),
	city VARCHAR(20),
	therapist VARCHAR(100)
);
INSERT INTO therapist_directory(name, gender, insurance,
new_patients, speciality_one,speciality_two, speciality_three,
license, phone)
VALUES ('Flora Martinez', 'Female', 'Yes', 'Yes', 'OCD',
'Phobias', 'Anxiety', 'MD', '8495776489'),
('Andy James', 'Male', 'Yes', 'No', 'Depression', 'Anxiety',
'PTSD', 'PhD', '2340894766'),
('Hannah Myers', 'Female', 'No', 'Yes', 'Anxiety',
'Schizophrenia', 'Bipolar', 'MD', '9907846574'),
('Jane Huang', 'Female', 'Yes', 'Yes', 'Depression', 'Anxiety',
'Bipolar', 'MD', '4507856797'),
('April Adams', 'Female', 'No', 'Yes', 'OCD', 'Anxiety', 'PTSD',
'MFT', '4507856797'),
('Jon Schaffer', 'Male', 'Yes', 'No', 'BPD', 'Bipolar',
'Depression', 'PhD', '9907846574'),
('Shauna West','Female', 'Yes', 'Yes', 'ADHD', 'Anxiety', 'OCD',
'MD', '8495776480'),
('Juan Angelo', 'Male', 'No', 'Yes', 'Schizophrenia', 'Bipolar',
'Depression', 'MD', '4507856797'),
('Christie Yang', 'Female', 'Yes', 'Yes', 'Autism', 'ADHD',
'OCD', 'PhD', '4507856796'),
('Annika Neusler', 'Female', 'Yes', 'No', 'Addiction',
'Depression', 'PTSD', 'MFT', '9907846575'),
('Simone Anderson', 'Female', 'No', 'No', 'Schizophrenia',
'Depression', 'PTSD', 'MD', '8304498765'),
('Ted Nyguen', 'Male', 'Yes', 'Yes', 'ADHD', 'Anxiety',
'Phobias', 'PhD', '4301239990'),
('Valentino Rossi', 'Male', 'Yes', 'Yes', 'Autism', 'Anxiety',
'Depression', 'MD', '8304498765'),
('Jessica Armer', 'Female', 'No', 'Yes', 'PTSD', 'Bipolar',
'Depression', 'MD', '3330456612'),
('Sid Michaels', 'Female', 'Yes', 'Yes', 'OCD', 'Phobia',
'Anxiety', 'MFT', '4301239997'),
('Yen Waters', 'Male', 'Yes', 'Yes', 'Anxiety', 'Depression',
'ADHD', 'PhD', '4507856796'),
('Ru Izaelia', 'Female', 'No', 'Yes', 'Bipolar', 'BPD',
'Phobias', 'MD', '4301239990'),
('Vishal Rao', 'Male', 'Yes', 'Yes', 'Depression',
'Schizophrenia', 'Anxiety', 'MD', '7305557894'),
('Lana John', 'Female', 'Yes', 'Yes', 'Anxiety', 'Phobias',
'OCD', 'MFT', '7305557894'),
('Izzie Geralt', 'Female', 'Yes', 'Yes', 'Depression',
'Addiction', 'Anxiety', 'MD', '4301239990');

INSERT INTO hospitals(hospital_name, state, city, therapist)
VALUES ('Van Holsen Community Hospital', 'CA', 'San Francisco',
'Flora Martinez'),
('Clear Water Services', 'CA', 'San Diego', 'Andy James'),
('Imagery Health', 'CA', 'Sacramento', 'Hannah Myers'),
('Blue Cross Clinic', 'CA', 'Los Angeles', 'Jane Huang'),
('Blue Cross Clinic', 'CA', 'Los Angeles', 'April Adams'),
```

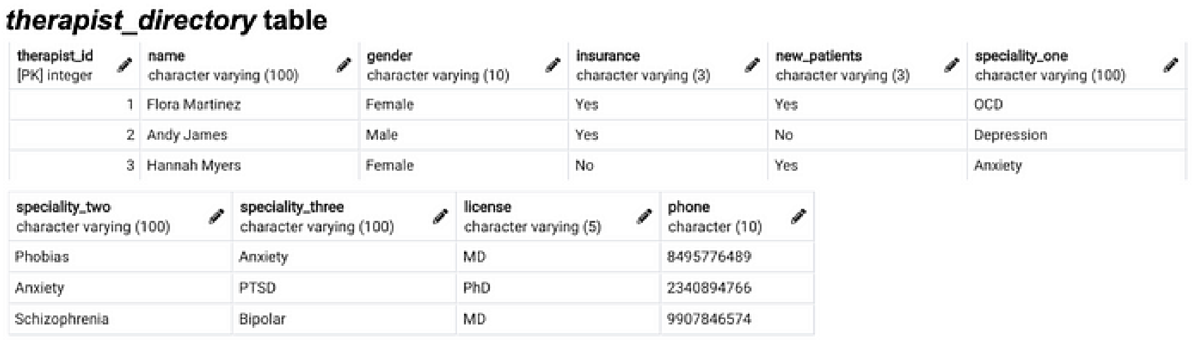
A Complete Guide to Database Normalization in SQL

```
('Imagery Health', 'CA', 'Sacramento', 'Jon Schaffer'),
('Van Holsen Community Hospital', 'CA', 'Long Beach', 'Shauna
West'),
('Blue Cross Clinic', 'CA', 'Santa Barbara', 'Juan Angelo'),
('Blue Cross Clinic', 'CA', 'San Francisco', 'Christie Yang'),
('Imagery Health', 'CA', 'Auburn', 'Annika Neusler'),
('Holistic Health Services', 'CA', 'Santa Barbara', 'Simone
Anderson'),
('Open Clinic', 'CA', 'San Jose', 'Ted Nyguen'),
('Holistic Health Services', 'CA', 'Santa Barbara', 'Valentino
Rossi'),
('Clark Jamison Hospitals', 'CA', 'Fresno', 'Jessica Armer'),
('Open Clinic', 'CA', 'Oakland', 'Sid Michaels'),
('Blue Cross Clinic', 'CA', 'San Francisco', 'Yen Waters'),
('Open Clinic', 'CA', 'San Jose', 'Ru Izaelia'),
('Clear Minds Community', 'CA', 'Sacramento', 'Vishal Rao'),
('Clear Minds Community', 'CA', 'Sacramento', 'Lana John'),
('Open Clinic', 'CA', 'San Jose', 'Izzie Geralt');
```

From the tables, we can see we have an array of variables pertaining to the therapists, where they work, what they specialize in, and how we might contact them.

Here’s a glance into the available data itself before we get started. Always double check your values before decomposition- *don’t assume you know what’s in a table by column name alone.*

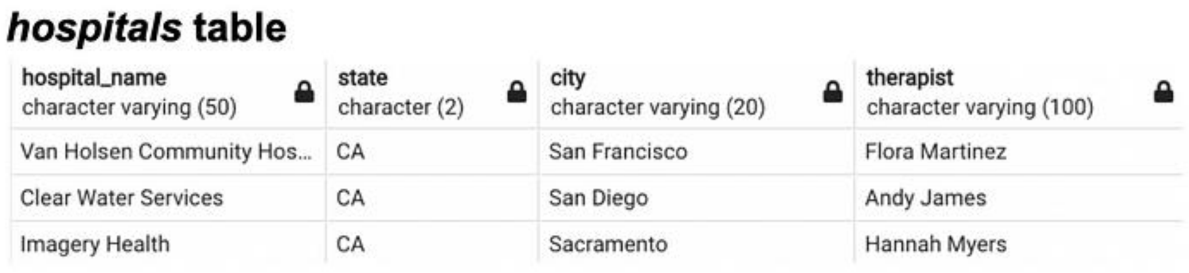
A Complete Guide to Database Normalization in SQL



therapist_directory **table**

therapist_id [PK] integer	name character varying (100)	gender character varying (10)	insurance character varying (3)	new_patients character varying (3)	speciality_one character varying (100)
1	Flora Martinez	Female	Yes	Yes	OCD
2	Andy James	Male	Yes	No	Depression
3	Hannah Myers	Female	No	Yes	Anxiety

speciality_two character varying (100)	speciality_three character varying (100)	license character varying (5)	phone character (10)
Phobias	Anxiety	MD	8495776489
Anxiety	PTSD	PhD	2340894766
Schizophrenia	Bipolar	MD	9907846574



hospitals **table**

hospital_name character varying (50)	state character (2)	city character varying (20)	therapist character varying (100)
Van Holsen Community Hos...	CA	San Francisco	Flora Martinez
Clear Water Services	CA	San Diego	Andy James
Imagery Health	CA	Sacramento	Hannah Myers

Please also keep in mind that **all synthetic data utilized here is for demonstrative purposes only** and does not accurately represent Californian hospitals, therapist demographics, or typical dataset size (as you probably already know, SQL is often used for big data projects, not tables with 20 total rows).

An Initial Query

With the database shown above, your company would like you to run a query:

Determine the number of therapists in Northern California that specialize in mood disorders and, of these therapists, how many are currently accepting new patients.

However, after looking at the tables, you can tell attempting accurate queries could prove to be a challenge. A relationship between the two tables has yet to be established and there appears to be redundant information. You'll have to do some decomposition prior to running the requested query.

1NF

Recall that the first step to normalization (1NF) concerns proper row identification and grouping data correctly.

Let's first remedy the relationship between the two tables: *therapist_directory* (parent table) and *hospitals* (child table). *therapist_directory* already has a primary key ("therapist_id"), but *hospitals* is lacking both primary and foreign keys. We can add an incrementally increasing primary key to *hospitals* with IDENTITY/SEQUENCE and PRIMARY KEY specifications:

```
ALTER TABLE hospitals ADD COLUMN hospital_id IDENTITY(1,1)
PRIMARY KEY;
```

Adding a foreign key will require pulling information from the *therapist_directory* table to accurately link it with *hospitals*. Although the "therapist" column in the *hospitals* table has each therapist's name, this isn't the ideal foreign key since, as you will see in the next steps, we will update the corresponding "name" column in *therapist_directory*. Instead, let's add the therapists' IDs to *hospitals* for consistency's sake.

```
UPDATE hospital h
SET therapist_id = td.therapist_id
FROM therapist_directory td
WHERE td.name = h.therapist;
```

Now we can update the included "therapist_id" column in hospitals to indicate that it is a foreign key, accurately linking the tables in the database. Consequently dropping the "therapist" column from *hospitals* will also ensure the data in the table is relevant specifically to the therapists' places of work.

```
ALTER TABLE hospitals
    ADD CONSTRAINT fk_therapist_directory
    FOREIGN KEY (therapist_id)
    REFERENCES therapist_directory(therapist_id);
```

Now let's also address redundant information present in the tables.

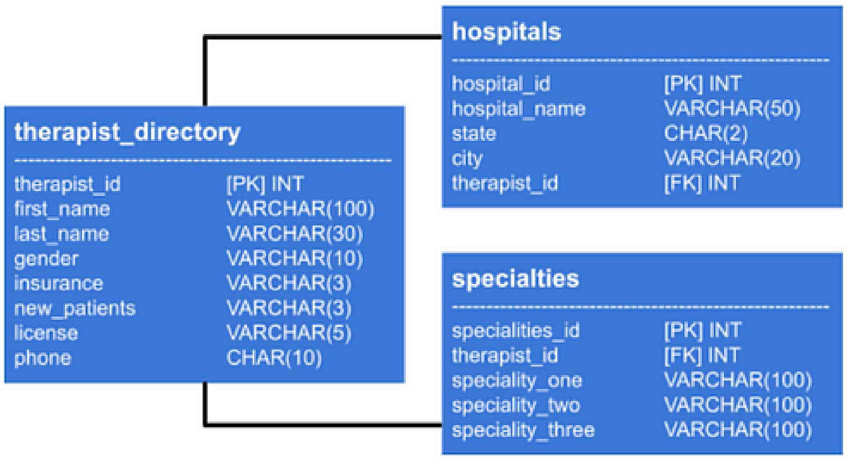
After referring to the data again, we see that, in order to keep the data in its most reduced form, we should split *therapist_directory*'s "name" column into first and last names columns with the use of SUBSTRING().

```
/* make new last_name column */

ALTER TABLE therapist_directory ADD COLUMN last_name
VARCHAR(30);

/* add last name values to last_name */

UPDATE therapist_directory
SET last_name = SUBSTRING(name, POSITION(' ' IN name)+1,
LENGTH(name));

/* update name column to first_name */

ALTER TABLE therapist_directory
RENAME COLUMN name TO first_name;



/* remove last name substring from first_name */

UPDATE therapist_directory
SET first_name = SUBSTRING(first_name, 1, POSITION(' ' IN
first_name)-1);
```

There are also repeating groups of data in the *therapist_directory* table ("speciality_one", "speciality_two" and "speciality_three"). We'll move these variables to their own table. *Don't forget to add a primary and foreign key to the new table!*

```
CREATE TABLE IF NOT EXISTS specialties(
    specialties_key serial PRIMARY KEY,
    speciality_one VARCHAR(100),
    speciality_two VARCHAR(100),
    speciality_three VARCHAR(100),
    therapist_id INTEGER,
    CONSTRAINT fk_therapist
        FOREIGN KEY(therapist_id)
        REFERENCES therapist_directory(therapist_id));

/* inserting data into new specialities table */

INSERT INTO specialties(therapist_id, speciality_one,
speciality_two, speciality_three)

SELECT therapist_id, speciality_one, speciality_two,
speciality_three
FROM therapist_directory;

/* dropping speciality columns from therapist_directory */

ALTER TABLE therapist_directory
DROP COLUMN speciality_one,
DROP COLUMN speciality_two,
DROP COLUMN speciality_three;
```

Let’s take a look at the database schema now.



Everything looks good so far. Time to move onto 2NF.

2NF

Recall that second normal form involves removing data unrelated to the primary key and establishing foreign keys to fortify table relationships.

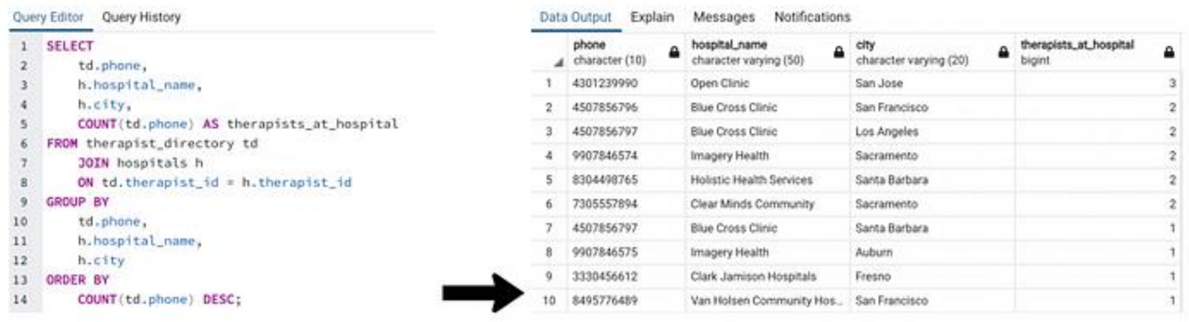
Since all of our tables currently have foreign keys (hurrah!), we can focus on identifying unrelated data.

We have three columns in *therapist_directory* that don't match up well with the "therapist_id" primary key: **insurance, new_patients, and phone**. "insurance" and "new_patients" don't identify who each therapist is. Rather, each points to visit specifications to consider when choosing a therapist.

With this in mind, let's move these two variables to their own table called *visit_specifications*. We can do so by again utilizing the CREATE TABLE syntax. And, remember, as long as the tables are connected with a one-to-one relationship, it's okay if a column doubles as the primary and the foreign key.

```
CREATE TABLE IF NOT EXISTS visit_specifications(
    therapist_id INTEGER PRIMARY KEY,
    insurance VARCHAR(3) CHECK(insurance IN ('Yes', 'No')),
    new_patients VARCHAR(3) CHECK(insurance IN ('Yes', 'No')),
    CONSTRAINT fk_visits
        FOREIGN KEY(therapist_id)
        REFERENCES therapist_directory(therapist_id));

/* inserting data into new table */

INSERT INTO visit_specifications(therapist_id, insurance,
new_patients)

SELECT therapist_id, insurance, new_patients
FROM therapist_directory;
```

Great, now we only have one unrelated column in *therapist_directory* left: **phone**. Looking closely, we can see the numbers listed are actually the hospitals' phone numbers rather than each therapist's personal contact info.



```
SELECT
    td.phone,
    h.hospital_name,
    h.city,
    COUNT(td.phone) AS therapists_at_hospital
FROM therapist_directory td
    JOIN hospitals h
    ON td.therapist_id = h.therapist_id
GROUP BY
    td.phone,
    h.hospital_name,
    h.city
ORDER BY
    COUNT(td.phone) DESC;
```

	phone character (10)	hospital_name character varying (50)	city character varying (20)	therapists_at_hospital bigint
1	4301239990	Open Clinic	San Jose	3
2	4507856796	Blue Cross Clinic	San Francisco	2
3	4507856797	Blue Cross Clinic	Los Angeles	2
4	9907846574	Imagery Health	Sacramento	2
5	8304498765	Holistic Health Services	Santa Barbara	2
6	7305557894	Clear Minds Community	Sacramento	2
7	4507856797	Blue Cross Clinic	Santa Barbara	1
8	9907846575	Imagery Health	Auburn	1
9	3330456612	Clark Jamison Hospitals	Fresno	1
10	8495776489	Van Holsen Community Hos...	San Francisco	1

We could go ahead and add the phone numbers to the more appropriate *hospitals* table, but this modification would be in violation of 3NF! Let's move onto third normal form to full detail why this is the case.

A Complete Guide to Database Normalization in SQL

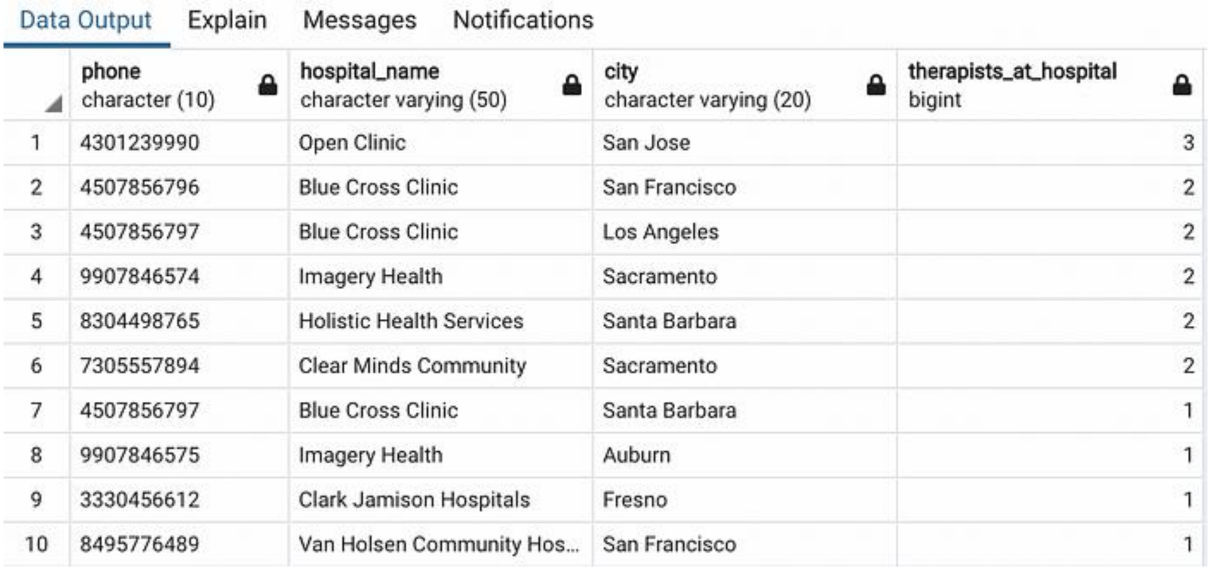
3NF

Recall that third normal form involves removing **transitively dependent** columns.

…What does that actually mean? 🤨

Let's walk step-by-step with the "phone" column example to make sense of this requirement.

As mentioned, we could add the "phone" column to the *hospitals* table to better match the "hospital_id" primary key. However, the phone number values would then depend on both "hospital_id" *and* the city that the hospital is in. Let's take another look at the data returned from our previous query:



Data Output | Explain | Messages | Notifications

	phone character (10)	hospital_name character varying (50)	city character varying (20)	therapists_at_hospital bigint
1	4301239990	Open Clinic	San Jose	3
2	4507856796	Blue Cross Clinic	San Francisco	2
3	4507856797	Blue Cross Clinic	Los Angeles	2
4	9907846574	Imagery Health	Sacramento	2
5	8304498765	Holistic Health Services	Santa Barbara	2
6	7305557894	Clear Minds Community	Sacramento	2
7	4507856797	Blue Cross Clinic	Santa Barbara	1
8	9907846575	Imagery Health	Auburn	1
9	3330456612	Clark Jamison Hospitals	Fresno	1
10	8495776489	Van Holsen Community Hos...	San Francisco	1

We can clearly see that therapists who work at the same hospital in the same city all have the same phone number listed (*for example, there are 2 therapists who work at Blue Cross Clinic in San Francisco*), but therapists who work at the same hospital in varying locations have different phone numbers (*there are 2 other therapists who also work at Blue Cross Clinic but, because they are located in Los Angeles, they have a different phone number than the two San Francisco-based Blue Cross Clinic therapists*).

This means that phone numbers depend on both the hospital (primary key) and the city the hospital is located in. Putting the "phone" values into their own table rather than the *hospitals* table safeguards against accidentally changing a therapist's city without also changing the phone number.

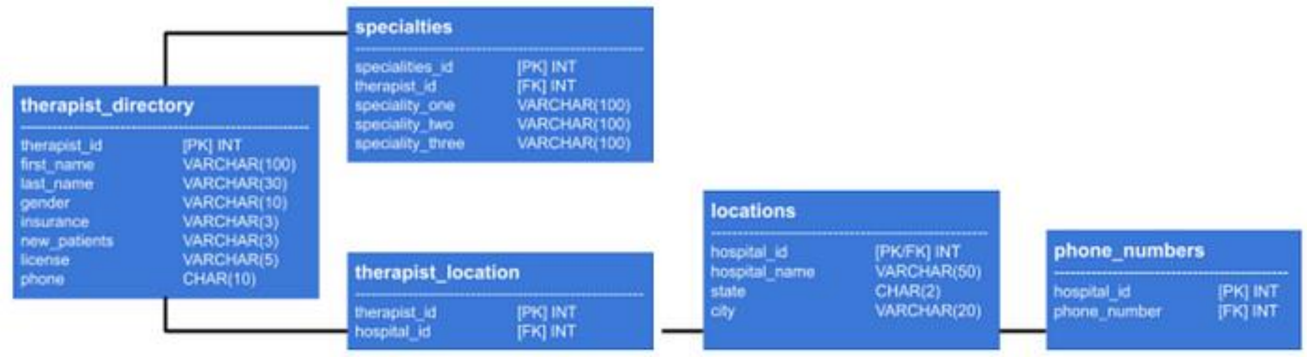
This is the basis of transitive dependence: *column values change based on the primary key and other columns also in the table.*

To circumvent this breach of data integrity, we can create three tables: one to establish a key for each location-specific hospital, one for phone numbers according to hospital id, and one for each therapist's corresponding hospital id.

```
/* creating locations table */
CREATE TABLE IF NOT EXISTS locations(
      hospital_id SERIAL PRIMARY KEY,
      hospital_name VARCHAR(50),
        state CHAR(2),
        city VARCHAR(20));

INSERT INTO locations(hospital_name, state, city)
VALUES
        ('Van Holsen Community Hospital', 'CA', 'San Francisco'),
        ('Clear Water Services', 'CA', 'San Diego'),
        ('Imagery Health', 'CA', 'Sacramento'),
        ('Blue Cross Clinic', 'CA', 'Los Angeles'),
        ('Van Holsen Community Hospital', 'CA', 'Long Beach'),
        ('Blue Cross Clinic', 'CA', 'Santa Barbara'),
        ('Blue Cross Clinic', 'CA', 'San Francisco'),
        ('Imagery Health', 'CA', 'Auburn'),
        ('Holistic Health Services', 'CA', 'Santa Barbara'),
        ('Open Clinic', 'CA', 'San Jose'),
        ('Clark Jamison Hospitals', 'CA', 'Fresno'),
        ('Open Clinic', 'CA', 'Oakland'),
        ('Clear Minds Community', 'CA', 'Sacramento');

/* creating new table for therapist_location */
CREATE TABLE IF NOT EXISTS therapist_location(
        therapist_id INTEGER PRIMARY KEY,
        hospital_id INTEGER,
        CONSTRAINT fk_therapist_hospital
       FOREIGN KEY(hospital_id)
          REFERENCES locations(hospital_id));

/* inserting data into new therapist_location table */
INSERT INTO therapist_location(therapist_id, hospital_id)
SELECT DISTINCT td.therapist_id, l.hospital_id
FROM therapist_directory td
JOIN hospitals h ON td.therapist_id = h.therapist_id
JOIN locations l ON h.hospital_name = l.hospital_name AND h.city
= l.city;

/* creating phone numbers table */
CREATE TABLE IF NOT EXISTS phone_numbers(
        hospital_id INTEGER PRIMARY KEY,
        phone_number CHAR(10),
        CONSTRAINT fk_phones FOREIGN KEY(hospital_id)
        REFERENCES locations(hospital_id));

/*inserting data into phone numbers table */
INSERT INTO phone_numbers(hospital_id, phone_number)
```

A Complete Guide to Database Normalization in SQL

```
SELECT DISTINCT tl.hospital_id, td.phone
FROM therapist_directory td
JOIN therapist_location tl ON td.therapist_id = tl.therapist_id;

/* dropping hospitals table */
DROP TABLE hospitals
```



Awesome, now our columns depend only on their respective primary key! This concludes our normalization steps 🥳

The Initial Query Revisited

Now that the database meets 1NF, 2NF, and 3NF standards, you can revisit the query your company requested. Here's a refresher just in case:

SQL used to look at distribution of phone numbers across each hospital and therapist

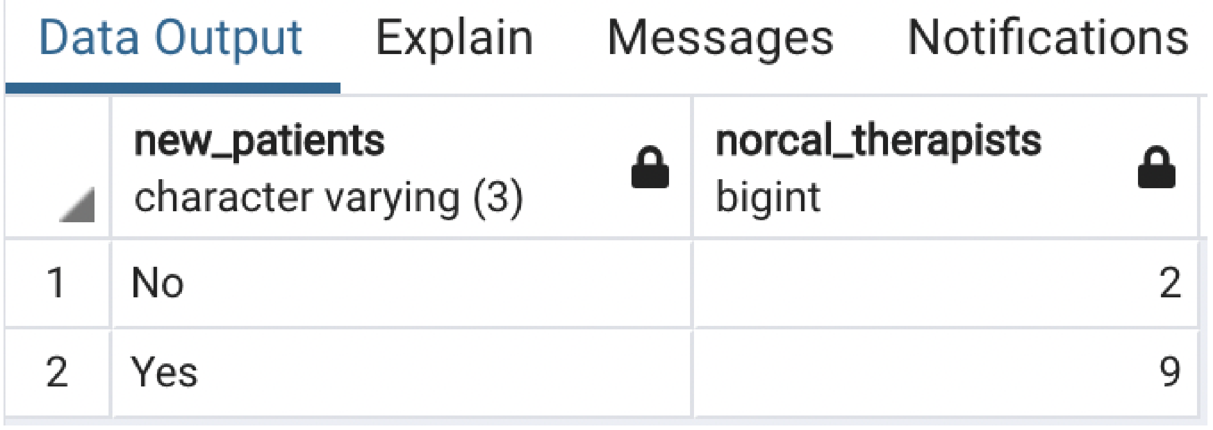
```
SELECT td.phone, h.hospital_name, h.city,
COUNT(td.phone) AS therapists_at_hospital
FROM therapist_directory td
JOIN hospitals h ON td.therapist_id = h.therapist_id
GROUP BY td.phone,h.hospital_name,h.city
ORDER BY COUNT(td.phone) DESC;
```

A Complete Guide to Database Normalization in SQL

Determine the number of therapists in Northern California that specialize in mood disorders and, of these therapists, how many are currently accepting new patients.

Assuming that mood disorders include “Anxiety”, “Depression”, and “Bipolar”, you can use the following query without creeping doubts about data integrity:

```
SELECT
 sub.new_patients,
 COUNT(therapist_id) AS norcal_therapists
FROM
 (SELECT s.therapist_id, s.speciality_one, s.speciality_two,
s.speciality_three, td.new_patients
  FROM specialties s
   JOIN therapist_directory td ON s.therapist_id =
td.therapist_id
   JOIN therapist_location tl ON td.therapist_id =
tl.therapist_id
   JOIN locations l ON tl.hospital_id = l.hospital_id
  WHERE l.city ~ '(San Francisco|Oakland|San
Jose|Sacramento|Auburn)') sub
WHERE
 speciality_one ~ '(Anxiety|Depression|Bipolar)'
 OR speciality_two ~ '(Anxiety|Depression|Bipolar)'
 OR speciality_three ~ '(Anxiety|Depression|Bipolar)'
GROUP BY sub.new_patients;
```



	new_patients character varying (3)	norcal_therapists bigint
1	No	2
2	Yes	9

According to the results, there are 11 therapists in Northern California that specialize in mood disorders and, of these 11, nine are currently accepting new patients!

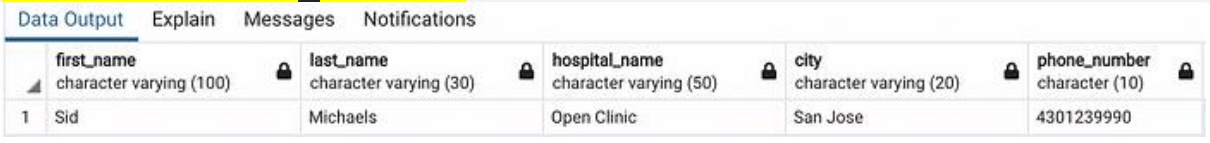
Beyond general analysis like this, a normalized database with non-transitively dependent columns likewise allows us to complete **data manipulation queries** with confidence. If your company had instead asked you to change the city (but not the hospital) a therapist works in, you could easily do so with the use of a single UPDATE statement without worrying about accidentally failing to simultaneously update the phone number.

```
UPDATE
    therapist_location
SET
    hospital_id = (SELECT hospital_id
                   FROM locations
                   WHERE hospital_name = 'Open Clinic'
                       AND city = 'San Jose')
WHERE
    therapist_id = 15;
```

The above query changes Sid Michael's (therapist_id = 15) hospital location to San Jose rather than Oakland. Because her hospital id is now up-to-date in the *therapist_location* table, we don't need to change Sid's contact info by hand with another UPDATE query. Instead, Sid's contact info is automatically updated according to her new hospital id. We can double-check this with a JOIN statement.

```
SELECT
    td.first_name,
    td.last_name,
```

```sql
    l.hospital_name,
    l.city pn.phone_number
FROM
    therapist_directory td
    JOIN
        therapist_location tl
        ON td.therapist_id = tl.therapist_id
    JOIN
        locations l
        ON l.hospital_id = tl.hospital_id
    JOIN
        phone_numbers pn
        ON tl.hospital_id = pn.hospital_id
WHERE
    td.therapist_id = 15;
```



	first_name character varying (100)	last_name character varying (30)	hospital_name character varying (50)	city character varying (20)	phone_number character (10)
1	Sid	Michaels	Open Clinic	San Jose	4301239990

A Brief Closing Message

Decomposing a database can be as simple as 1(NF), 2(NF), 3(NF)! For more insight into the mock "Northern_California_Therapists" database and the individual SQL queries synthesized for this article,

Happy normalizing! 🤖