Homework Problem Set 10: Data Normalization

# Overview

In this lab, we will explore how to normalize data and then migrate data to new tables in the process.

## Learning Objectives

Upon completion of the lab, you should be able to:

* Identify data dependencies and normal forms.
* Resolve data dependencies by placing data in a higher normal form.
* Move data into new normalized tables.

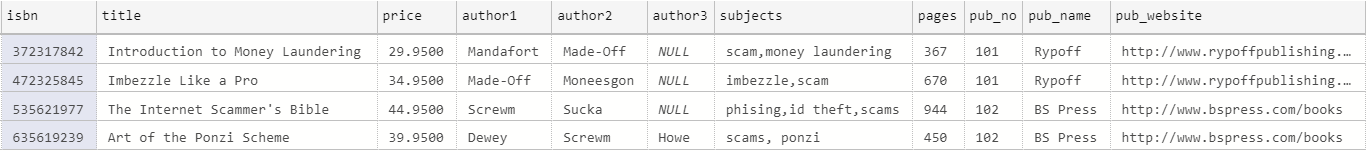
## What You Will Need

To complete this lab, you will need the learn-databases environment up and running, specifically:

* Microsoft SQL Server DBMS.
* Provision the **demo** and **cheepwebhosting** databases using the database provisioner application <https://localhost:5000>.
* Azure Data Studio connected to SQL Server with an open query window.
* **Fudgenbooks.sql** file from where you got this lab.
* Please review the first lab if you require assistance with these tools.

# Walkthrough

In this walkthrough, we will normalize the Fudgenbooks database. Execute the **fudgenbooks.sql** script to create the table **fudgenbooks** in the **demo** database. The **isbn** column is the primary key of this table.



To review, for the data normalization process, we must perform these steps:

1. Resolve any columns not dependent on the key (if they exist), then
2. Resolve any partial key dependencies (if they exist), then
3. Resolve any transitive dependencies (if they exist), and finally
4. Add the foreign key constraints.

In general, each of the three resolution processes are similar. It involves splitting the data in the original table by placing the columns with dependencies into a new table. How the new tables are created will depend on the type of dependency. We figure this out by first writing the SELECT statement to get the desired data.

Next, we write a migration script to create the table and insert the data:

1. Write a DROP TABLE before the SELECT as part of the up/down process.
2. Use the SELECT statement you wrote but add the INTO clause to create a new table and insert the query output.
3. ALTER the table after the SELECT to add the primary key (we want entity integrity).
4. Run a SELECT on the new table to verify the data are correct.

## Step 1: Resolve Columns With No Dependency on the Key

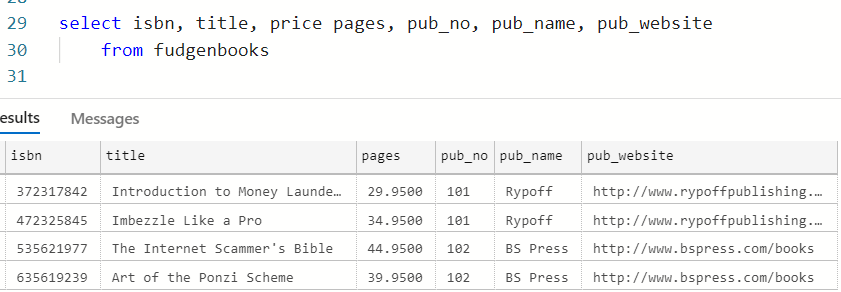
In the fudgenbooks example, you cannot get a single atomic value for **author** for the given key **isbn.** Likewise, you cannot get a single **subject**, either. This is because these columns contain multivalued attributes, which are not desirable in a relational table because it is not trivial to query the data.

Because both **subjects** and **authors** are hidden many-to-many relationships, we must resolve each to a bridge table and lookup table. When we are finished, we should have five tables:

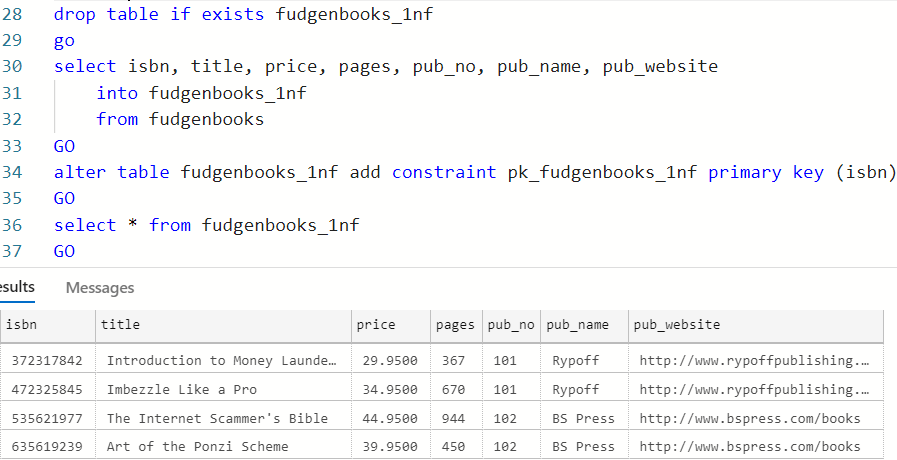
1. The 1NF version of fudgenbooks with the no key dependencies removed
2. A lookup table of unique authors
3. A bridge table connecting each **isbn** to its many **author**(s)
4. A lookup table of unique subjects
5. A bridge table connecting each **isbn** to its many **subject**(s)

### Step 1.1: fudgenbooks\_1nf

The 1NF version of fudgenbooks is simple:



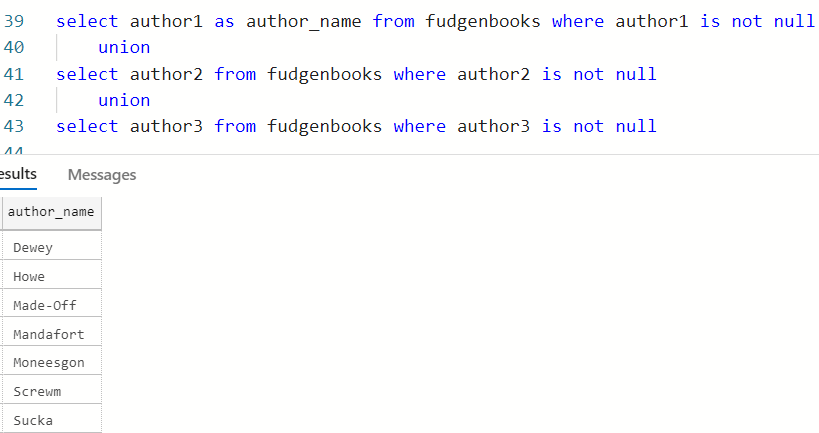
Then we write our migration script using the four-step resolution process from above (drop, make table, add PK constraint, select to verify).



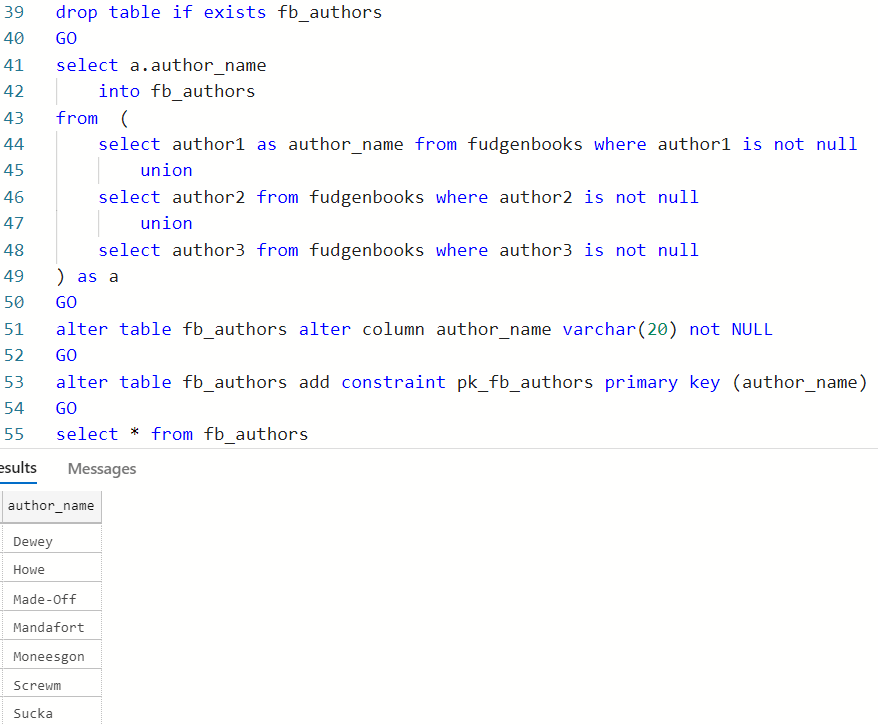
The **INTO** clause on line 31 will create a new table from the query output. This is the key to migrating the normalized data.

### Step 1.2: fb\_authors Lookup Table

To create the lookup table, we must combine the unique values from columns **author1**, **author2**, and **author3**. Here, a UNION query does the trick. This is the common approach to use when there are multiple columns.



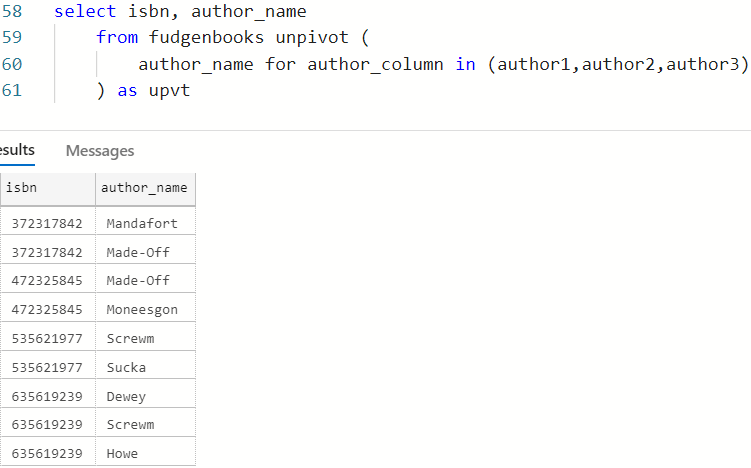
With the desired output, we then turn this into a migration script with our four-step process once more:



Notice in this migration script we had to alter the **author\_name** column, setting it to **not null**. This is required because the table created from the INTO clause allows null on all columns except the original primary key, **isbn**.

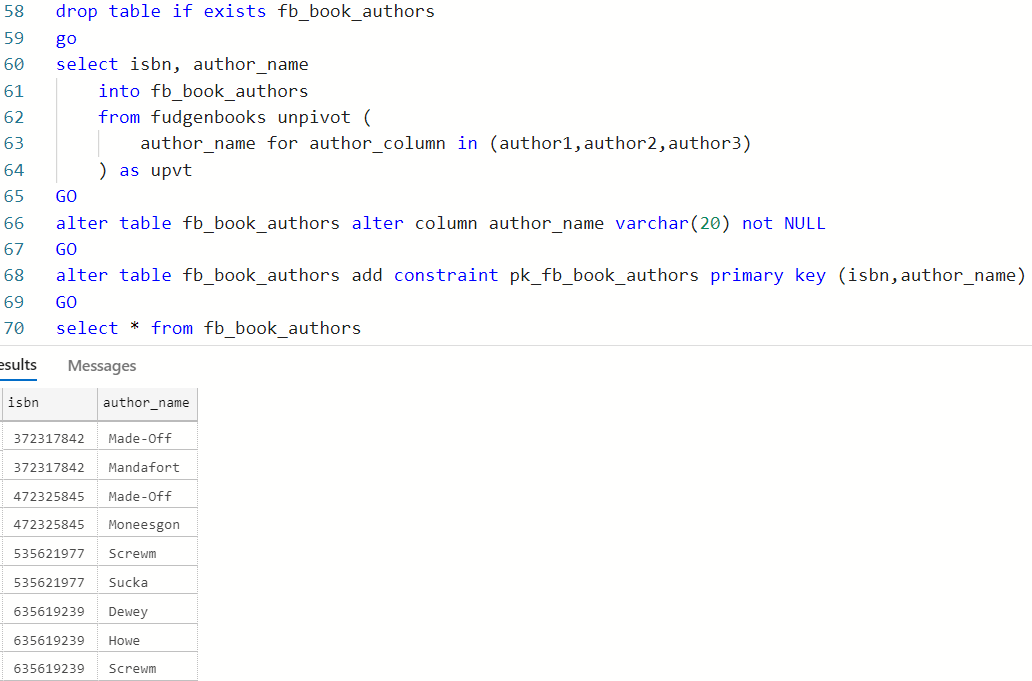
### Step 1.3: fb\_book\_authors Bridge Table

In the last step of the resolution of the **authors** columns, we must create the bridge table, assigning each **isbn** and **author\_name** a row in the table. When the values are in multiple columns, we use the UNPIVOT clause to build the bridge table:



Notice this data still line up with the original data (three authors of book with isbn 635619239, for example), only now the data are easier to query.

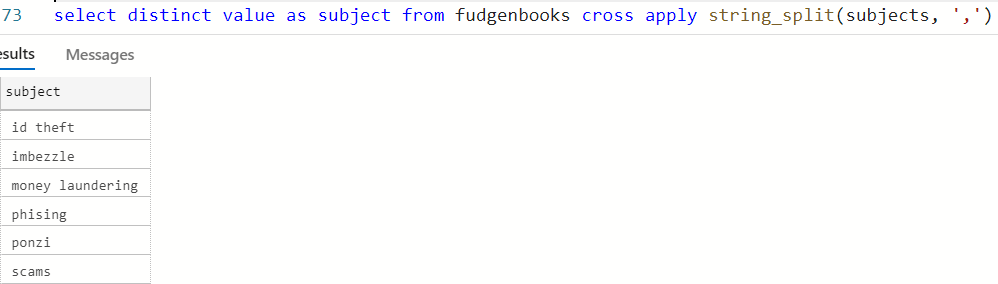
Next, we transform this query into a migration script:



Note how we used a composite primary key in this migration script. This makes sense because the **fb\_book\_authors** table is a bridge table.

### Step 1.4: fb\_subjects Lookup Table

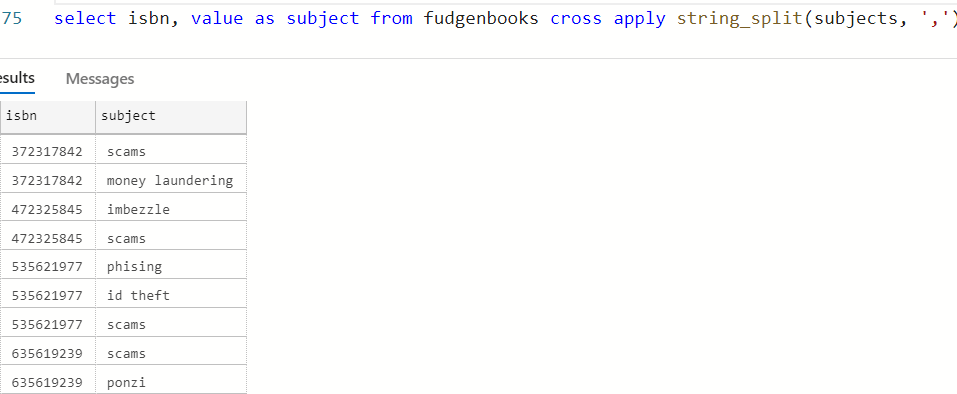
Because the **subjects** column is a multivalued single column, we use the STRING\_SPLIT function to help us extract the values. We also need distinct to pare down the number of unique values in the lookup table:



It is left to the reader to turn this into a migration script that creates the table **fb\_subjects**, populated with data and having the appropriate primary key set.

### Step 1.5: fb\_book\_subjects Bridge Table

Here is the SQL to for the bridge table, which is similar to the lookup table.



It is left to the reader to turn this into a migration script that creates the **fb\_book\_subjects** table.

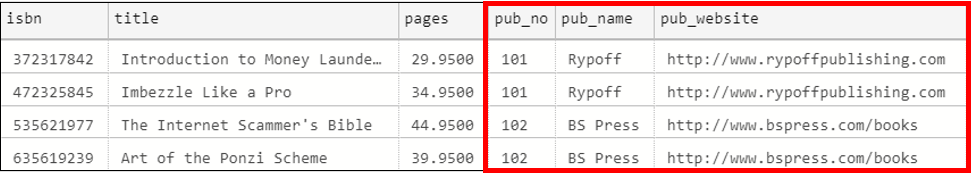
When we are finished, we should have five tables, and with all columns key dependent, we are in first normal form (1NF).

## Step 2: Resolve Any Partial Key Dependencies

We can skip this step because partial dependencies apply only to composite primary keys. The only composite primary keys are in the bridge tables. At this point there are no partial key dependencies. We are now in second normal form (2NF)

## Step 3: Resolve Any Transitive dependencies

In this final step, we must resolve any transitive dependencies. These occur when a non-key acts as a key for other non-key columns. Observe:

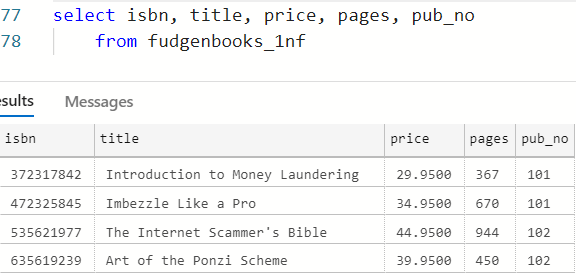


The **pub\_no** (publisher number) column acts as a key for the **pub\_name** and **pub\_website** columns. In actuality, this table has four books and two publishers. The first two books were published by Rypoff Publishing, and the last two books were published by B. S. Press. Transitive dependencies are hidden 1-M relationships, like this one that we have between publisher and book. To resolve, we must:

1. Create a 3NF version of fudgenbooks\_1nf. We will call this table **fb\_books** with the transitive dependencies removed. We must leave **pub\_no** in the table as the FK. This is the “many” side table of the hidden 1-M relationship.
2. The table **fb\_publishers** should contain the **pub\_no** as primary key and the transitively dependent columns. This table is the “one” side of this hidden 1-M relationship.

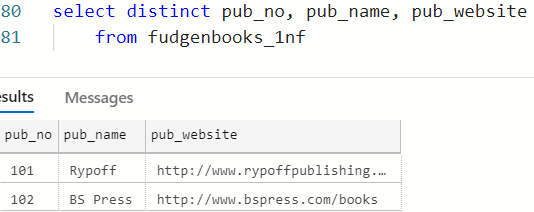
### Step 3.1: fb\_books Table From the Original Table

First, remove the transitive dependencies from the **fudgenbooks\_1nf**  table to create the table **fb\_books**.



The migration script for the table **fb\_books** is left to the reader to complete.

### Step 3.2: fb\_publishers Table From Transitive Dependencies

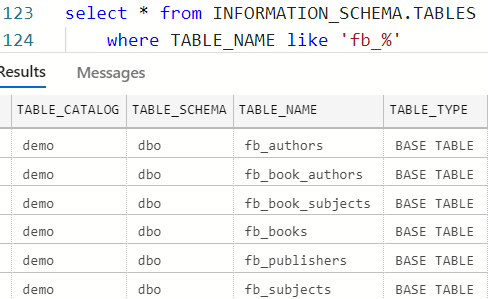


The migration script for the table **fb\_publishers** is left to the reader to complete.

At this point, there are no transitive dependencies, so we are now in third normal form (3NF).

## Step 4: Add the Foreign Keys

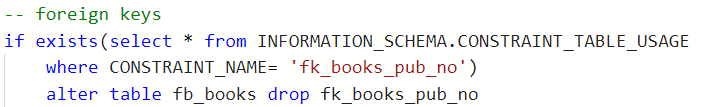
At this point, our tables are in third normal form. Our normalized model is complete, and so now we should reintroduce our foreign keys back into the model. Here are our tables:



Left to the reader, write an up/down script to add the following foreign keys:

|  |  |  |  |
| --- | --- | --- | --- |
| **Table** | **Column** | **FK Name** | **References** |
| fb\_book\_authors | isbn | fk\_book\_authors\_isbn | fb\_books(isbn) |
| fb\_book\_authors | author\_name | fk\_book\_authors\_author\_name | fb\_authors(author\_name) |
| fb\_book\_subjects | isbn | fk\_book\_subjects\_isbn | fb\_books(isbn) |
| fb\_book\_subjects | subject | fk\_book\_subjects\_subject | fb\_subjects(subject) |
| fb\_books | pub\_no | fk\_books\_pub\_no | fb\_publishers(pub\_no) |

Your script should alter the tables, add the FKs at the bottom of the script, and drop the foreign keys (if they exist) before you start the migration. Code like this, which soft-drops the FK, should appear at the top, before any migrations.



(Repeat for each foreign key.)

When you are finished, you should have a single script to normalize and migrate the data to new tables!

# Questions

Answer these questions using the problem set submission template. For any screen shots provided, please follow the guidelines for submitting a screen shot.

1. Provide a screen shot of your working migrations for Steps 1.4, 3.1, and 3.2 in the walkthrough.
2. Provide a screen shot of your adding foreign keys in Step 4 and a separate screen shot of your code to drop the foreign keys.

Normalize the **xyz\_consulting** database. You can get this script in the same place you got this lab.

1. Provide a screen shot of your migration scripts (if any) to 1NF.
2. Provide a screen shot of your migration scripts (if any) to 2NF.
3. Provide a screen shot of your migration scripts (if any) to 3NF.
4. Provide a list of tables in 3NF.
5. Add all foreign keys back to the new model.