

Role Playing Dimension of the Year

written by DaltonRuer | November 16, 2022



The ballots have been tabulated and the **2022 Dorksi** for Role Playing Dimension of the Year goes to the “Physicians Table.”

The CEO, CIO, CMO, CNO of the fictitious, soap opera based General Hospital Health System all shared the same quote – “The Physicians Table is literally the lifeblood of our health system.”

The Chief Data Officer elaborated a bit more by saying “I’m impressed daily with all of the roles that the physicians table plays in our organization. It’s versatility and range is truly mind boggling. Every day of the year it just keeps putting in the work. It represents the Admitting Physician when needed. Yet can represent a Surgeon just moments later with no time to prepare. Then come right back after lunch and take on the role of a Discharging Physician.”

Role Playing Dimensions

Truth be told there is no **Dorksi** Award and quotes from fictitious people don’t really count. But the flexibility of certain dimensional tables does need to be celebrated. Or at least discussed from a **modeling** perspective.

In it’s naturally occuring state a [Role Playing Dimension](#) is simply a table with a primary key and some other fields like

any other table. However, most dimensions are clearly known and only represent one thing. For example a Patient table would have some primary key like Master_Patient_ID that would be referred to in other tables. Always representing the patient. If the Department table is referred to, everyone knows it is the Department. Simple Foreign Key to Primary Key relationships that are referred to in fact tables.

The physician table, however, isn't quite so simple. In a healthcare system things are driven from an Encounters (fact) table that would identify which patient came in, at what time, for what reason. Lot's of those simple foreign key to primary key relationships. Within that encounter table instead of containing a Physician ID you would find fields like Admitting_Physician_ID, Admitting_Physician_ID and Discharging_Physician_ID. No doubt they are foreign keys. But to what? Hence, the phrase Role Playing Dimension.

Reporting "experts" would know how to perform SQL joins to the table as needed and would then rename columns in their SQL. But the goal of good Data Modeling is to make it easy for these Role Playing Dimensions to be more readily identified for self-service users.

Modeling Options in Qlik Cloud Data Integration (QCDI)

Within QCDI there are 3 distinct options you have when it comes to dealing with these role playing dimensions.

1. You can use a Transformation step to create **datasets** for each role. Thus an Admitting_Physician dataset, an Attending_Physician dataset and a Discharging_Physician dataset.
2. When building a Data Mart you can simply "denormalize" the information from the Physician table and flatten it out so that your Encounter table simply has the columns

of information you choose with the appropriate names like "Attending_Physician_FullName, Attending_Physician_LicensingState etc."

3. When building a Data Mart you can ... this is crazy ... literally create the role playing dimensional **datasets** on the fly.

If you have any data modeling experience under your belt then you have probably flipped out at the notion that we would create separate **tables** that all contained the same information. I totally agree. But if you reread #1 again be sure to take note that I didn't say **table** I said **DATASET**. You see those term differences are key as you model your data flows within QCDI. Tables are always tables. But datasets can be materialized or not materialized. Meaning they can just be **views**. And by default that's exactly what will happen and I will show you what is actually created below.

If you are thinking through the options you probably realize that option 3 seems just like option 1. They are very similar with the biggest difference being option 1 is available to be queried from by anything at any time. It isn't dependant on constructing data marts, and those transformed datasets would be available to all data marts you build. I can ensure you that our encounters table isn't the only fact table that will need to access that hard working physician table.

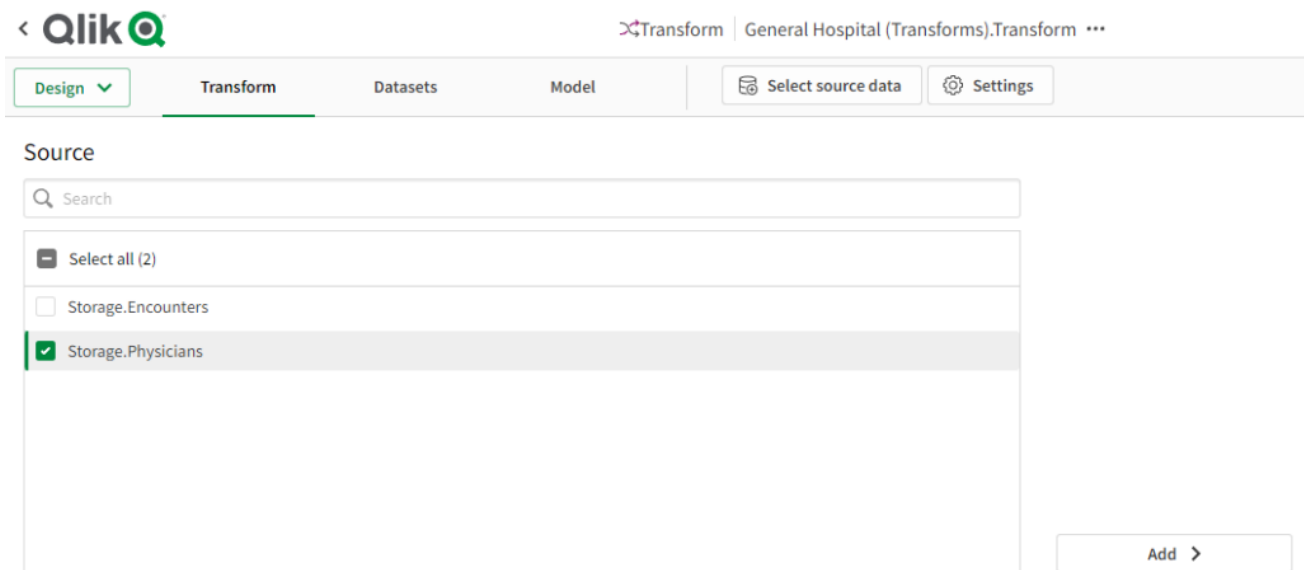
Below I will document, with pretty pictures for you, each of the options I described above so you understand and can implement each of these modeling options for your role playing dimensions. They may not win my 2022 Dorksi Award, but surely they are as critical to your business as the physician table is for healthcare.

Option 1: Transforming the Physician

Table

As you ingest data from your source system(s), like the Physicians table, they will be in the exact same structures that they start in. Our goal with this option is to make life easier down stream. We want to transform our Physicians Table from a single dataset playing many roles, into the multiple Role Playing Datasets so that each one serves a singular purpose later on. Thus, easing the end users burden of understanding of finding the role playing fields by having matching column names for their primary key and foreign key relationships. We will use a Transform layer to accomplish that purpose.

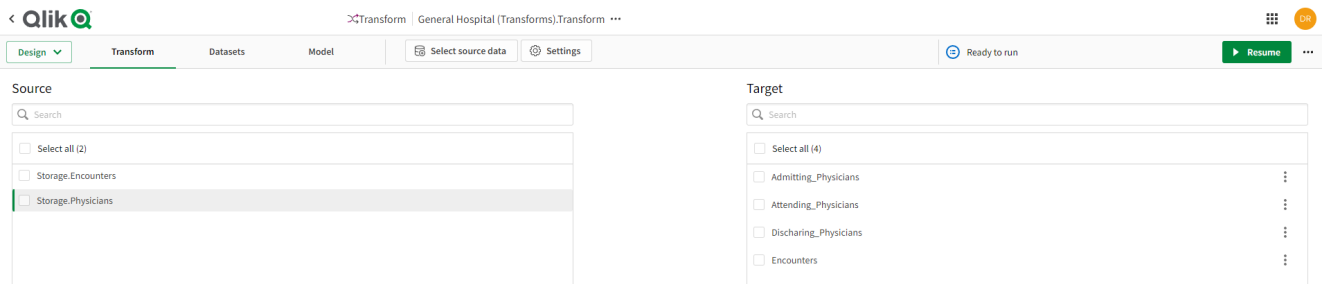
Step 1: The first step in your [Transofm](#) layer will be to select your table and press Add as many times as needed. In our case I needed to press Add 3 times.



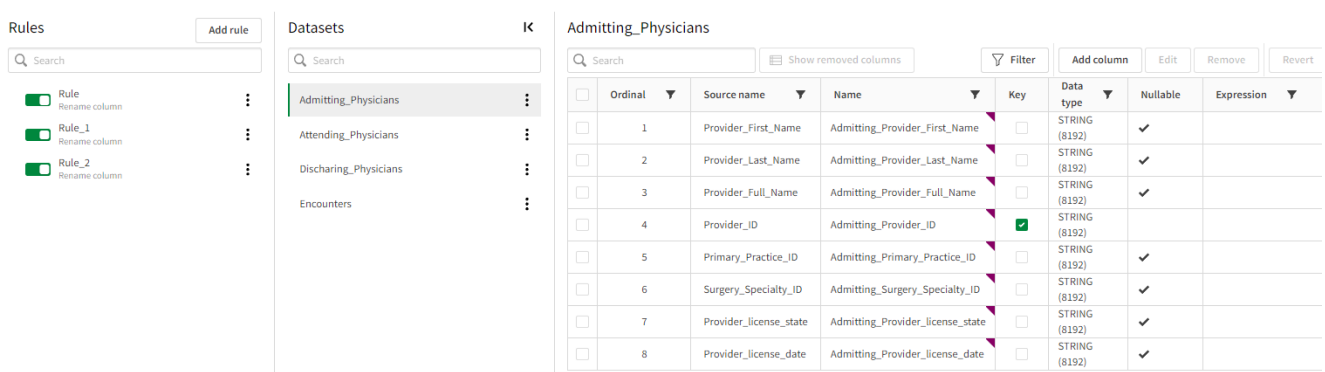
Step 2: For each of the “Physicians” datasets you will need to **Rename** them appropriately.



Step 3: Finally, in my Transform layer I ended up with 4 datasets. The Encounters dataset and the **3 role playing versions** of my Physicians table.



Step 4: The next step is to switch to the **Datasets** view so that you can then modify the names of the columns. “Provider_First_Name” is a perfect column to have in a versatile role playing table like Physicians, but it isn’t as perfect as “**Admitting_Provider_First_Name**” is when the field is in the Admitting_Physicians dataset. You can select each and every field and then press Edit to manually edit the field names. Totally appropriate if you only have a couple, or you want to really customize each and every field. In my case, as you will see on the left side of the screenshot below I defined Rules instead. One of the rules simply is “If the table name is Admitting_Physicians please prefix each column name with ‘Admitting_’.” As you can imagine, the other 2 rules do the same for the Attending and Discharging roles/datasets.



Step 5: Next step is to move to the **Model** view. Once there you will need to Add a relationship between your fact table and your role playing datasets.

QCDI provides a classic user interface to do that, so I went about identifying all 3 of the needed relationships.

Add relationship
X

Child dataset

Encounters

Parent dataset

Admitting_Physicians

Admitting_Provider_ID

Admitting_Provider_ID

Step 6: Then I simply went ahead and ran my transformation and voila I ended up with totally redundant physician tables in Snowflake. Just kidding. Like I mentioned when describing the options, these are **non materialized datasets** so QCDI creates them as Views. If I select anything from them, the views will simply execute select statements against the storage table version of the Physician table, but will alias the fields with the new names I assigned.

❖ transform
No Tables in this Schema
▼ Views
Admitting_Physicians
Admitting_Physicians__changes
Admitting_Physicians__history
Attending_Physicians
Attending_Physicians__changes
Attending_Physicians__history
Discharging_Physicians
Discharging_Physicians__changes
Discharging_Physicians__history
Encounters
Encounters__changes
Encounters__history
❖ transform__internal
▼ Tables
ASSET_STATE__636e6c9784cabd0...
▼ Views
Admitting_Physicians__whdr
Attending_Physicians__whdr
Discharging_Physicians__whdr
Encounters__whdr

Option 2: Denormalizing the Physician Table

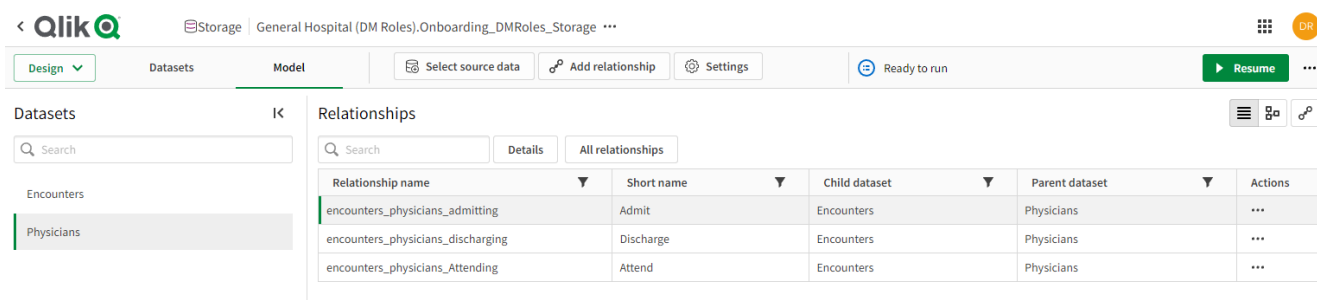
Your hardworking DBA's do such great work of removing redundancy in your data by normalizing it. While there are many benefits to their efforts, being able to quickly consume the data isn't one of them. It's bad enough when you need to walk all of the relationships for normal dimensions, but role playing dimensions like our Physician table means 3 times the work. Thus the term [Denormalizing](#) became a thing to simply unwind those relationships and flatten the data back into a table of redundant values. Then users can simply select from the Encounters table and see the information about the Admitting, Attending and Discharging physicians.

Options 1 and 3 are essentially the same but why do they even exist since this denormalizing seems so easy for the end users? The answer to that question will really come from your business/data stewards. Options 1 and 3 give them the ability to see ALL physicians and find the ones that have never been used as the Admitting, Attending or Discharging Physician for encounters. While Option 2 only provides the ability to see information about the ones that have been.

If you are thinking “I have 25 fields in my Physician table so I don’t want really want to denormalize all of them to the Encounters table.” You are in luck, because you are totally welcome to denormalize only the fields you choose.

If you are thinking “I would sure like to have the Physican Name denormalized to the Encounters table because 90% of the time that’s all that is needed. But I still need all of the fields available for the other use cases.” You are in luck, because Option 2 isn’t mutually exclusive from Options 1 and 3.

Step 1: The first step to denormalizing the physician table actually begins back at the Storage layer. You will need to create 3 relationships, just like you did in Option 1. The difference is that all 3 relationships will be between the Encounters table and the same Physicians table. **Be sure and note** that the Short Name you provide for the relationship is what will be used to prefix all of the fields from the Physicians table when it is denormalized.



Relationship name	Short name	Child dataset	Parent dataset	Actions
encounters_physicians_admitting	Admit	Encounters	Physicians	...
encounters_physicians_discharging	Discharge	Encounters	Physicians	...
encounters_physicians_Attending	Attend	Encounters	Physicians	...

Step 2: When you create your data mart layer and choose your fact table, any relationships that are defined will be

displayed and you can simply check them. In the following image you can see that all of three of the Physicians relationships are shown and I have checked them all.

Add star schema

Fact

Encounters

Name

Encounters

Description

Description

Related datasets to denormalize

▼ Encounters

☒ Physicians

☒ Physicians

☒ Physicians

Cancel

OK

Step 3: The only remaining step for you is to switch to the Datasets view and remove any of the denormalized fields that you really don't want flattened out to the Encounters table. In my case I only cared to flatten out the Provider's Full Name which is all that is needed 90% of the time. You can also Edit any of the columns to change/customize their Name.

Note that I purposely gave my relationships Short Names that would be obvious. In the screen shot you can see how it prefixed each field from the Physicians table with the Short Name I provided. Since the key names were Admitting, Attending and Discharging I didn't want to confuse you or have you

believe that is how it prefixed the field.

Qlik

Data mart | General Hospital (DM Roles).DataMart_Denormalized ...

Design | Data mart | Datasets | Select source data | Rules | Settings

Ready to run

Run

Rules

Add rule

Search

Encounters

No rules

You can create rules to perform transformations on multiple tables. For each rule you can choose condition, action, and scope. Rules will run in the order shown in list.

Add rule

Encounters

Search

Show removed columns

Add column | Edit | Remove | Revert

	Ordinal	Source name	Name	Key	Data type	Nullable	Expression	
<input type="checkbox"/>	11	Patient_Admitted_Flag	Patient_Admitted_Flag	<input type="checkbox"/>	STRING (8192)	✓		fx
<input type="checkbox"/>	12	Patient_Readmission_Flag	Patient_Readmission_Flag	<input type="checkbox"/>	STRING (8192)	✓		fx
<input type="checkbox"/>	13	Patient_Inpatient_Readmission_Flag	Patient_Inpatient_Readmission_Flag	<input type="checkbox"/>	STRING (8192)	✓		fx
<input checked="" type="checkbox"/>	14	Admit_Provider_First_Name	Admit_Provider_First_Name	<input type="checkbox"/>	STRING (8192)	✓		fx
<input type="checkbox"/>	15	Admit_Provider_Last_Name	Admit_Provider_Last_Name	<input type="checkbox"/>	STRING (8192)	✓		fx
<input checked="" type="checkbox"/>	16	Admit_Provider_Full_Name	Admit_Provider_Full_Name	<input type="checkbox"/>	STRING (8192)	✓		fx
<input checked="" type="checkbox"/>	17	Admit_Provider_ID	Admit_Provider_ID	<input type="checkbox"/>	STRING (8192)	✓		fx
<input checked="" type="checkbox"/>	18	Admit_Primary_Practice_ID	Admit_Primary_Practice_ID	<input type="checkbox"/>	STRING (8192)	✓		fx
<input checked="" type="checkbox"/>	19	Admit_Surgery_Specialty_ID	Admit_Surgery_Specialty_ID	<input type="checkbox"/>	STRING (8192)	✓		fx
<input checked="" type="checkbox"/>	20	Admit_Provider_license_state	Admit_Provider_license_state	<input type="checkbox"/>	STRING (8192)	✓		fx
<input checked="" type="checkbox"/>	21	Admit_Provider_license_date	Admit_Provider_license_date	<input type="checkbox"/>	STRING (8192)	✓		fx
<input checked="" type="checkbox"/>	22	Attend_Provider_First_Name	Attend_Provider_First_Name	<input type="checkbox"/>	STRING (8192)	✓		fx
<input checked="" type="checkbox"/>	23	Attend_Provider_Last_Name	Attend_Provider_Last_Name	<input type="checkbox"/>	STRING (8192)	✓		fx
<input type="checkbox"/>	24	Attend_Provider_Full_Name	Attend_Provider_Full_Name	<input type="checkbox"/>	STRING (8192)	✓		fx

Step 4: All that is left to do is Prepare the Data Mart and Run it. Now when any end user for any reason sees the Encounters dataset they will immediately see the Admitting Provider’s Last Name, the Attending Provider’s Full Name and the Discharging Provider’s Full Name.

datamart_denormalized

No Tables in this Schema

Views

Encounters

datamart_denormalized__internal

Encounters

Preview Data

Patient_Encounter_ID

VARCHAR(8192)

Patient_Admission_Datetime

VARCHAR(8192)

Admitting_Provider_ID

VARCHAR(8192)

Discharging_Provider_ID

VARCHAR(8192)

Attending_Provider_ID

VARCHAR(8192)

Patient_Discharge_Datetime

VARCHAR(8192)

Department_ID

VARCHAR(8192)

Hospital_Account_ID

VARCHAR(8192)

Patient_InICU_Flag

VARCHAR(8192)

Patient_Admitted_Flag

VARCHAR(8192)

Patient_Readmission_Flag

VARCHAR(8192)

Patient_Inpatient_Readmission...

VARCHAR(8192)

Admit_Provider_Last_Name

VARCHAR(8192)

Attend_Provider_Full_Name

VARCHAR(8192)

Discharge_Provider_Full_Name

VARCHAR(8192)

Option 3: Handling the Roles in your Data Marts

This option is very much like Option 1 with the exception being that it provides you a way to add the Role Playing relationships needed for specific Data Marts without having to have taken any prior actions.

Step 1: Similar to Option 2, the first step is to create the relationships between the fact table and your role playing dimension table. Refer to Step 1 for Option 2 to see how I created the 3 relationships between the Encounter and Physicians tables.

Step 2: After we have added our fact table, in our case the Encounters table, and whether or not I have implemented Option 2 to flatten some of the fields I press Add Dimension. With most Dimensions you would simply choose the dataset/dimension and press Ok. But with your Role Playing dimensions you will want to be sure to add the Name of the Role the table will be playing. Notice in the screen shot below I have said "And the Role the Physicians table will be playing this time is that of the Discharging_Physicians." You would need to do this for each Role that your dimension table will play. In my case I did the same thing for the Admitting_Physicians and the Attending_Physicians.

Add dimension

×

Most granular dataset

Physicians

▼

Name

Discharging_Physicians

Description

Description

History type

Type 1

Related datasets to denormalize

Physicians

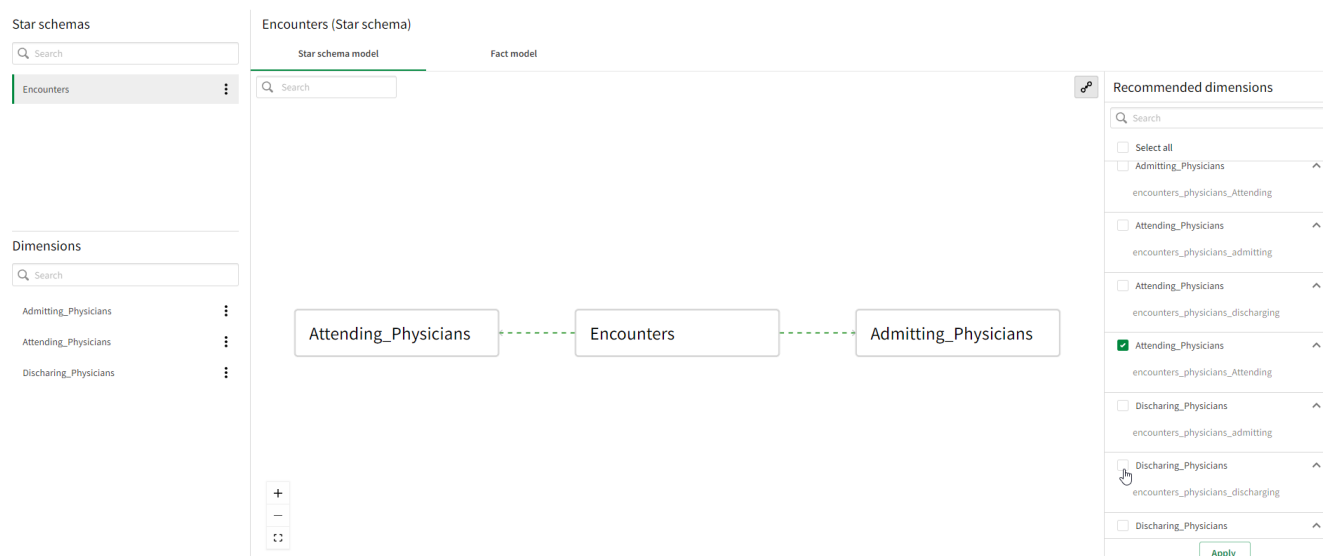
Cancel

OK

Step 3: The right side of the screen will show you the **Recommended dimensions** to relate to your fact table. Admittedly it seems a bit odd at first, A whole slew of crazy relationships. The key to accepting what is shown is to realize that just because you named your relationships in step 1, and just because you named your Role Playing Dimensions doesn't mean that QCDI knows which relates to each. After all, we could have named the Relationships Puppies, Flowers and Clouds if we wanted to. Just as we could have named our Role Playing Dimensions Monday, Tuesday and Friday.

With that said it's your job to point QCDI in the right direction and check the correct boxes that link together the right Role Playing Relationship with the right Role Playing Dimension we just created in Step 2. In the screenshot below I

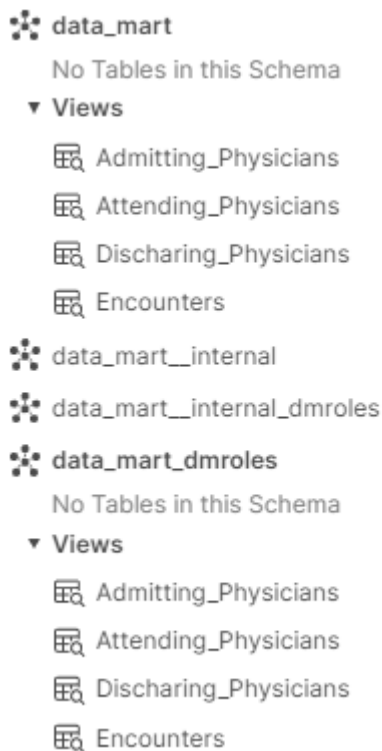
am about to check the final box to complete my 3 role playing relationships/dimensions. As you check the boxes you will notice that the fact table, in my case Encounters, has dashed lines. Once you complete yours, be sure to press the **Apply** button and those lines will be saved, and will become solid lines.



Step 4: That wasn't so bad was it? The last remaining step is to switch to the Datasets view and rename our fields. Because although there will be an Admitting_Physicians, Attending_Physicians and Discharging_Physicians tables all of the fields will still have the names given in the Physicians table. As with Option 1, Step 4, you are free to manually rename the fields or create rules.

In the screenshot below you can see that whether I used Option 1 to **transform** my Role Playing Physicians table and then built a data mart from those 3 individual dimensions, or use Option 3 to build the **Role Playing Dimensions** on the fly the result is the same. In terms of what end users would see when they look to consume the data. Option 1 provides the ability to refer to my transformed **Role Playing Datasets** if I create additional marts and it already handles the field renaming. But Option 3 provides the ability to create the **Role Playing Dimensions** on the fly if/when another role becomes available. "Oh look there is a Surgical Encounters table that we are

going to build a datamart for, and we need that Physicians table to become a Surgeons table.”



Summary

It's not everyday I get the chance to use an image I created with Artificial Intelligence like my “**Dorks*i***” for a post. May seem silly, but the truth is that there are so many other awards given out for roles that are far less impactful than a Role Playing Dimension.

In your business it might be the Customers table. Your Sales table might refer to it as Sold To Customer, Ship To Customer and Bill To Customer. Whatever your particular business is, I guarantee you have these incredibly versatile entities. Hopefully, I've done a good job of helping you see various ways you can model them with [Qlik Cloud Data Integration](#) as well as providing the tutorial for you to use to do it.