

[Get unlimited access](#)[Open in app](#)

Published in Whispering Data

You have **1** free member-only story left this month. [Upgrade for unlimited access.](#)



Paul Singman

[Follow](#)

Jun 12 · 6 min read · ✨ · 🎧 Listen



Save



5 Tips For a Tidy Data Warehouse

Spark joy in your data warehouse by following these data modeling best practices!



[Get unlimited access](#)[Open in app](#)

much value you get from your data. It can make the difference between a report or analysis taking a week of analyst time vs a day.

When data models are organized and clearly reflect business processes, it makes it easier for analysts to understand and make use of a company's data.

Let's take a look at the Do's and Dont's of data modeling:

1. **Categorize tables (and name them accordingly)**
2. **Don't shy away from convenience columns**
3. **Add a pre-built date dimension table**
4. **Add metadata columns to tables**
5. **Maintain a visual view of your table lineage**

Ready? Let's go for it...

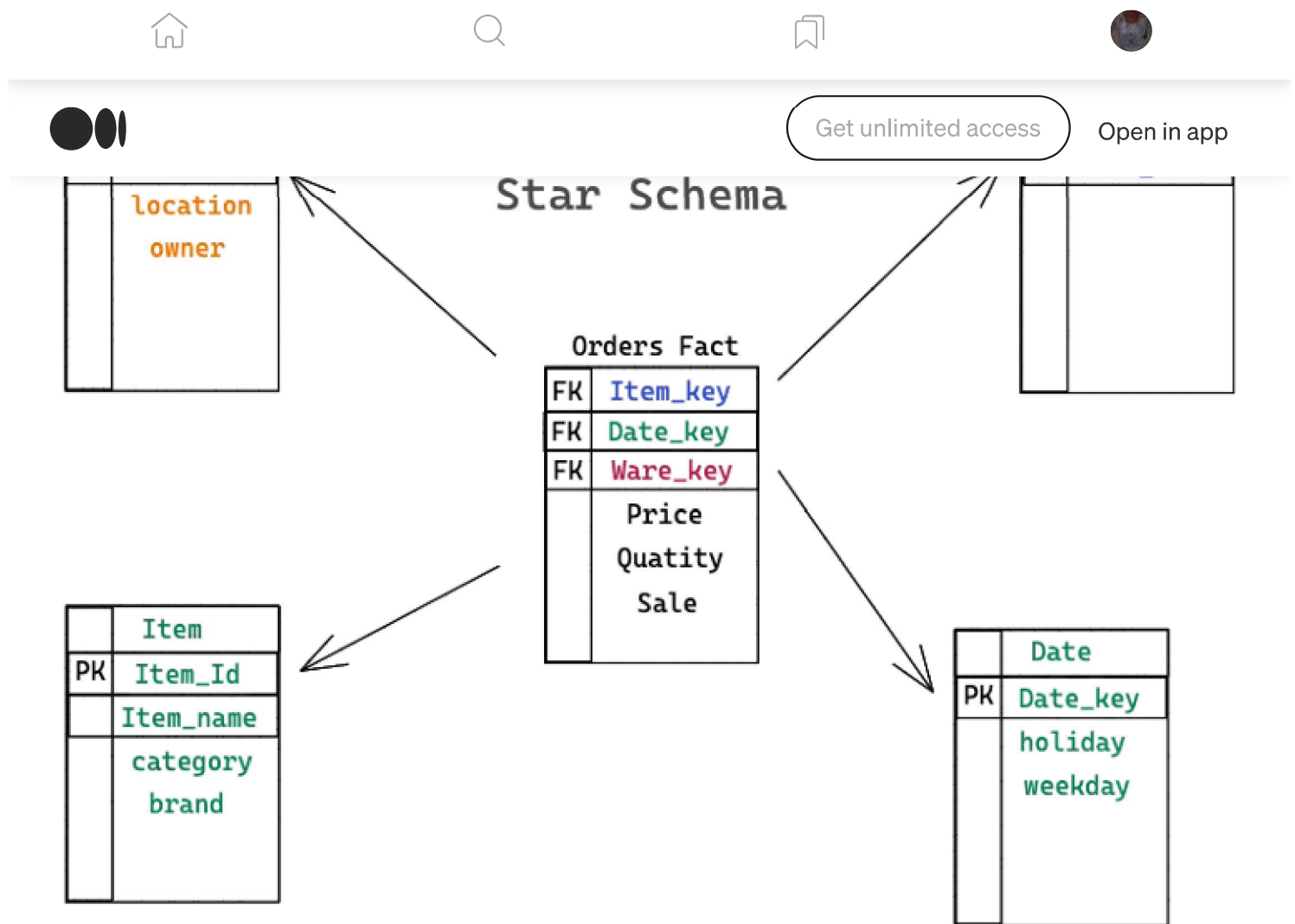
Tip #1: Categorize Tables (and name them accordingly)

Organization and structure are everything. Without it, we descend into a world of chaos.

Your data warehouse is no exception, particularly when it comes to the tables in it, of which there can grow to be many.

The most popular ways to organize a warehouse are with either the **star schema or snowflake schemas**. Both recommend making use of the concept of organizing tables into either **fact** or **dimension** tables.

Roughly speaking, fact tables contain a log of actions taken like "account created" or "order placed". They can easily grow to contain millions of rows, depending on how popular your business is.



Example tables in Star Schema.

Dimension tables, on the other hand, contain information about the user or entity that took the action, like a user's email address and other relevant attributes. They typically do not have as many rows as fact tables, but can be much wider.

When creating a new table ask yourself, *"Is this a fact table or a dimension table?"*

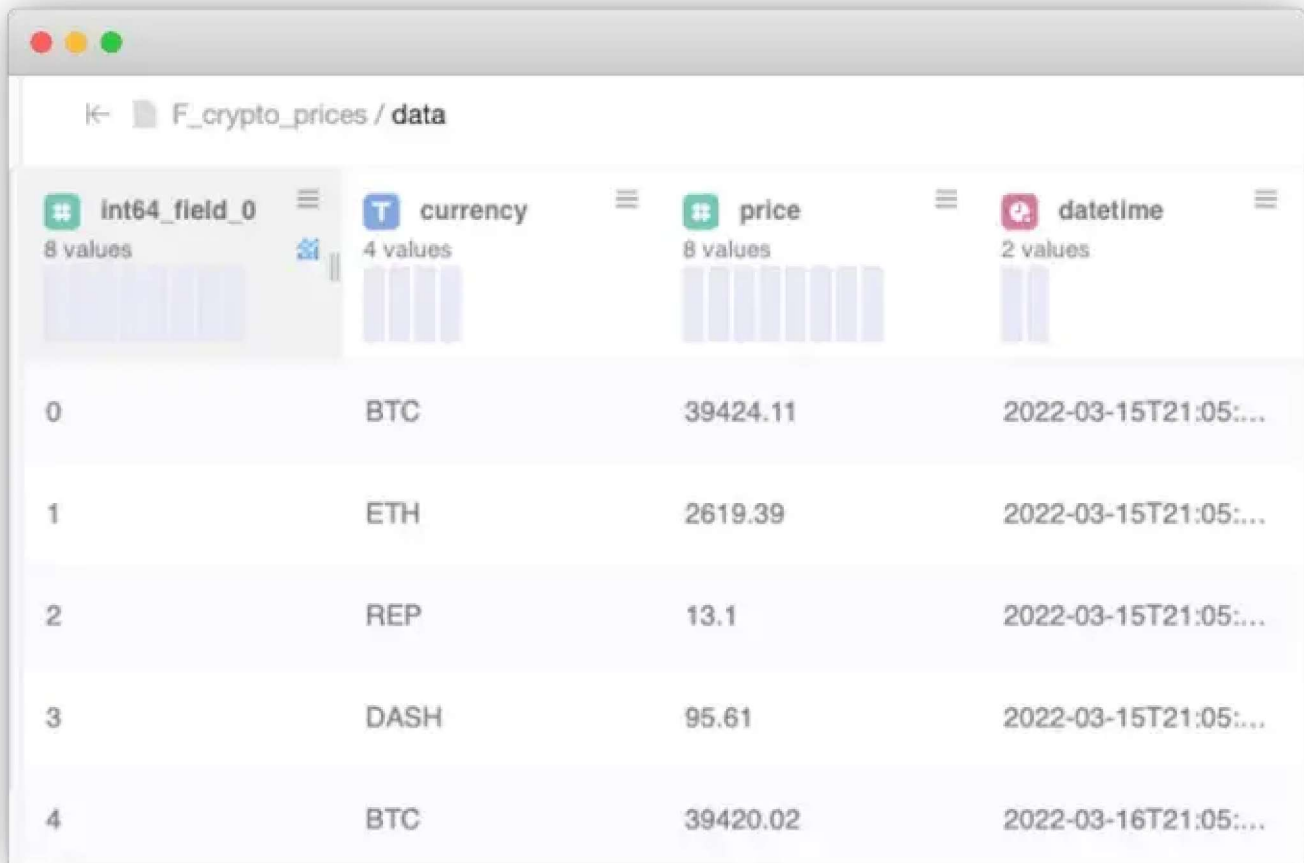
Then make it as easy as possible for other people to know as well by naming the table accordingly. For example, when creating a user dimension, name it **d_user** or **dim_user**. Similarly, prefix fact tables with **f_** or **fact_**.

This is a pattern I've seen adopted at several successful data organizations and helps ensure everyone on the team follows modeling best practices.

Tip #2: Don't Shy Away From Convenience Columns

[Get unlimited access](#)[Open in app](#)

As an example, take a look at the following fact table of crypto prices.



int64_field_0	currency	price	datetime
0	BTC	39424.11	2022-03-15T21:05:...
1	ETH	2619.39	2022-03-15T21:05:...
2	REP	13.1	2022-03-15T21:05:...
3	DASH	95.61	2022-03-15T21:05:...
4	BTC	39420.02	2022-03-16T21:05:...

First rows of `f_crypto_prices` fact table.

To perform a calculation of the daily price change for Bitcoin, we would write the following query:

```
1 with yest AS (  
2     SELECT currency, price  
3     FROM f_crypto_prices  
4     WHERE TRUNC(datetime, 'day') = '20220315')  
5 , today AS (  
6     SELECT currency, price  
7     FROM f_crypto_prices  
8     WHERE TRUNC(datetime, 'day') = '20220316')
```

[Get unlimited access](#)[Open in app](#)

13 FROM yest, today

14 WHERE currency = 'BTC'

f_crypto_prices_daily_change.sql hosted with ❤ by GitHub

[view raw](#)

What's annoying about writing this query is the date conversion of the datetime column using the `TRUNC` function. Doing it once is not a big deal, but when every query your write contains this clunky syntax, you'll start looking for a better way to do it.





That solution is to add a new column to the `f_crypto_prices` table that stores the date field in the exact format you use it in.


	int64_field_0	currency	price	datetime	datekey
0		BTC	39424.11	2022-03-15T21:05:...	20220315
1		ETH	2619.39	2022-03-15T21:05:...	20220315
2		REP	13.1	2022-03-15T21:05:...	20220315
3		DASH	95.61	2022-03-15T21:05:...	20220315
4		BTC	39420.02	2022-03-16T21:05:...	20220316

f_crypto_prices table with datekey column added.

Now we can rewrite the percent change query more simply as:


```
1 with yest AS (  
2     SELECT currency, price
```





[Get unlimited access](#) [Open in app](#)

```
8      WHERE datekey = '20220316')
9
10     SELECT
11         currency,
12         (today.price - yest.price) / yest.price * 100 AS pct_change
13     FROM yest, today
14     WHERE currency = 'BTC'
```

f_crypto_prices_daily_change_v2.sql hosted with  by GitHub [view raw](#)

Unless you are working with truly gigantic datasets, the cost of the extra column is negligible. And being able to write tidy, beautiful queries is priceless!

Tip #3: Add a Pre-calculated Date Dimension Table

When performing analyses over a data warehouse, perhaps no table is used more frequently than a date dimension, often named `d_date`.

Here’s what an example `d_date` table looks like:

Id	Year	Month	Day	FullDateAlt	DateKey	DateFullName	Fiscal Year	Fiscal Quarter	Calendar Quater	IsWeekDay	DayOfWeek	Month Name	Day of Week Name
1	2013	6	15	6/15/2013	20130615	15 June 2013	2013	4	2	0	6 June	Saturday	
2	2013	6	16	6/16/2013	20130616	16 June 2013	2013	4	2	0	6 June	Sunday	
3	2013	6	17	6/17/2013	20130617	17 June 2013	2013	4	2	1	1 June	Monday	
4	2013	6	18	6/18/2013	20130618	18 June 2013	2013	4	2	1	2 June	Tuesday	
5	2013	6	19	6/19/2013	20130619	19 June 2013	2013	4	2	1	3 June	Wednesday	
6	2013	6	20	6/20/2013	20130620	20 June 2013	2013	4	2	1	4 June	Thursday	
7	2013	6	21	6/21/2013	20130621	21 June 2013	2013	4	2	1	5 June	Friday	
8	2013	6	22	6/22/2013	20130622	22 June 2013	2013	4	2	0	6 June	Saturday	
9	2013	6	23	6/23/2013	20130623	23 June 2013	2013	4	2	0	6 June	Sunday	
10	2013	6	24	6/24/2013	20130624	24 June 2013	2013	4	2	1	1 June	Monday	
11	2013	6	25	6/25/2013	20130625	25 June 2013	2013	4	2	1	2 June	Tuesday	
12	2013	6	26	6/26/2013	20130626	26 June 2013	2013	4	2	1	3 June	Wednesday	
13	2013	6	27	6/27/2013	20130627	27 June 2013	2013	4	2	1	4 June	Thursday	
14	2013	6	28	6/28/2013	20130628	28 June 2013	2013	4	2	1	5 June	Friday	

Image source.

A date dimension contains a row for each day in the calendar, over many years in the past and future. And it has many, many columns describing important attributes of

[Get unlimited access](#)[Open in app](#)

In addition, using a date dimension avoids a well-known logical error that occurs when analyzing the frequency of an event that might not occur every day. Rather than using the dates in a fact table directly, begin writing your query by selecting all days from `d_date` and left join it to the relevant fact table to ensure no days are missing from the final results.

Using dimension tables in this way also helps your tables from growing too wide when following the advice in Tip #2.

Tip #4: Add Metadata Columns to Tables

There's no better place to save how and when a row in a table came to be than in the table itself.

It is a worthwhile use of storage space to include a few additional metadata columns in tables. The most common ones I see are called `updated_at` , `created_at` , and `updated_by` .

This lets you easily answer questions like “*show me all rows in the `d_users` table created in the past day*” or imported from a Salesforce sync with `Job ID #1435` .

The example `f_crypto_prices` table we've been using looks more complete with the metadata columns added:

index	currency	price	datetime	datekey	created_by	created_at	updated_at
0	BTC	39424.11	2022-03-15T21:05...	20220315	crypto_fetch 1234	2022-03-15T21:05...	2022-03-15T21:05...
1	ETH	2619.39	2022-03-15T21:05...	20220315	crypto_fetch 1234	2022-03-15T21:05...	2022-03-15T21:05...
2	REP	13.1	2022-03-15T21:05...	20220315	crypto_fetch 1234	2022-03-15T21:05...	2022-03-15T21:05...
3	DASH	96.61	2022-03-15T21:05...	20220315	crypto_fetch 1234	2022-03-15T21:05...	2022-03-15T21:05...
4	BTC	39420.02	2022-03-16T21:05...	20220316	crypto_fetch 1245	2022-03-16T21:05...	2022-03-16T21:05...
5	ETH	2669.39	2022-03-16T21:05...	20220316	crypto_fetch 1245	2022-03-16T21:05...	2022-03-16T21:05...

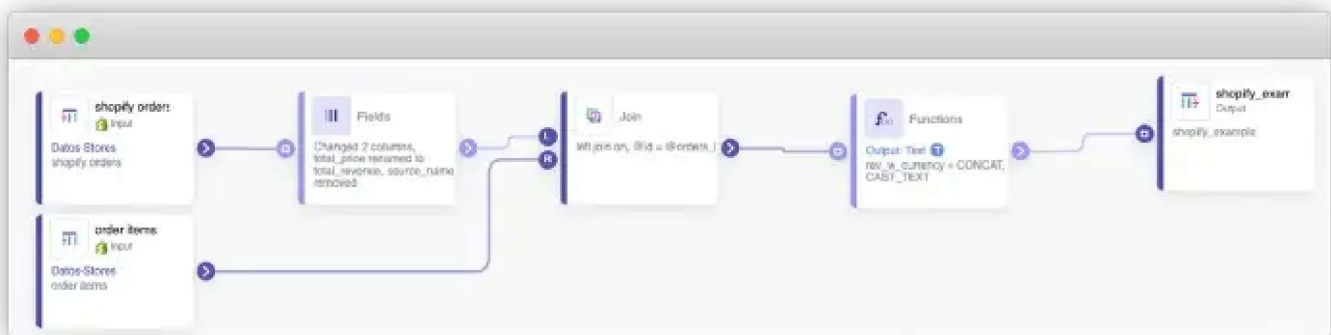
f_crypto_prices with new metadata columns!

Having these fields present in every fact and dimension table in your data warehouse makes it nicer for all to use.

Tip #5: Maintain a Visual View of Your Table Lineage

No one — and I mean no one — enjoys the process of debugging a data error by tracing through SQL code to identify table dependencies.

The process is less mentally taxing when references between tables can be viewed visually. This allows you to easily examine which upstream tables could have been the original source of a funny datapoint.



Example Lineage view.

In the above screenshot we see the table structure as visualized over an example

[Get unlimited access](#)[Open in app](#)

order_items tables.

However you define your data models, I would make sure it is done with a tool that provides such visualizations out of the box.

Final Thoughts

Having a logical and intuitive data model in your data warehouse is one of the most important things you can do to improve the efficiency of everyone on your data team.

I hope these tips are helpful whether you are starting a data model from scratch or looking to improve your existing tables!

Sign up for Whispering Data

By Whispering Data

Whispering Data is a publication for all the data & productivity secrets you wish you knew years ago! [Take a look.](#)

Emails will be sent to kalongboniface97@gmail.com. [Not you?](#)

[Get this newsletter](#)