DATA ENGINEERING

Data Modeling for Analytics

Dimensional design, slowly changing dimensions and the

Houssam Kodad

PDF · DATAFORGE BOOKS

Questions about this sample or the full edition: **support@dataforgebooks.com**

CONTENTS

Table of Contents

CHAPTER 01

Why Modelling Still Matters in the Cloud

Most warehouse pain does not come from the tooling. It comes from the model. When analysts cannot find the right table, when two dashboards disagree about revenue, when a "simple" question takes a day to answer — the root cause is almost always a schema that grew without design. This book treats data modelling as the craft it is, and this chapter makes the case for why that craft still matters in an era of effectively unlimited compute.

We will look at why cheap cloud compute changed which modelling problems are worth solving, why the analyst — not the source system — is the user you design for, and how to recognise the symptoms of a model that is quietly costing your organisation time and trust. We close by introducing the two ideas everything else in the book rests on: the fact table and its grain.

The promise of good modelling is concrete. A well-modelled warehouse lets a competent analyst answer a new question correctly in minutes, with confidence that the number matches every other number in the business. That is not a luxury; for a data team it is the entire point.

Cheap Compute, Expensive Confusion

When compute was scarce, modelling was justified mainly by performance: normalise, index, and optimise so queries would finish at all. The cloud warehouse removed that pressure — you can throw compute at a badly shaped query and it will still return an answer. Some teams concluded that modelling no longer matters. They were half right: performance modelling matters less, but clarity modelling matters more than ever.

The expensive resource today is not CPU, it is human understanding. A well-modelled warehouse lets an analyst answer a question correctly in minutes. A pile of denormalised exports lets them answer it three different ways, two of them wrong, and then spend an afternoon working out which to trust. The cost moved from the machine to the people — and people are far more expensive than query time.

The Analyst Is Your Real User

A dimensional model is a user interface, and its users are the analysts and decision-makers who query it. Like any interface, it can be intuitive or hostile. Good models speak the language of the business: a sales fact surrounded by customer, product, date and store dimensions maps directly onto the questions people actually ask, so writing the query feels like asking the question.

This reframing has practical consequences throughout the book. You design grain around the questions being asked, not the shape of the source table. You name columns the way the business names things, not the way an upstream application happened to. You make the common path easy and the rare path possible. When you model for the analyst rather than for the source system, adoption and trust follow almost automatically.

Facts and Dimensions

Dimensional modelling divides the world into two kinds of table. Facts record events or measurements — an order, a payment, a shipment — and hold the numeric values you want to aggregate, plus foreign keys to context. Dimensions hold that context: who, what, where, when. A fact answers "how much"; the dimensions answer every other question you might group or filter by.

The shape this produces is the star schema: a central fact table surrounded by dimension tables, joined by keys. It is deliberately denormalised compared to an application database, because its job is different. An application optimises for writing one row correctly; an analytical model optimises for reading and aggregating millions of rows comprehensibly. Recognising that these are different goals is the first real insight of the discipline.

```
-- A fact surrounded by dimensions: the star schema
select
    d.calendar_month,
    p.category,
    sum(f.amount_eur) as revenue
from   fct_sales f
join   dim_date    d on d.date_key    = f.date_key
join   dim_product p on p.product_key = f.product_key
group  by 1, 2
```

Declaring the Grain

The single most important decision in modelling a fact table is its grain: the precise definition of what one row represents. "One row per order line" or "one row per shipment per day" — the grain is a sentence you should be able to say out loud before you write any SQL. Everything else, from which dimensions attach to which measures are additive, follows from it.

The cardinal sin is mixing grains in one table: storing order-level totals next to line-level details, so that summing a column double-counts. When a fact table has a clear, uniform grain, aggregations are trustworthy by construction. When it does not, every query becomes a small investigation into whether the number is real. Declare the grain, write it in the documentation, and never violate it.

Slowly Changing Dimensions

Dimensions are not static; a customer moves city, a product is re-categorised, a salesperson changes team. How you handle that change is one of the defining choices of a dimensional model. The Type 1 approach simply overwrites the old value, keeping only the current truth. It is simple, but it silently rewrites history — last year's sales suddenly attribute to this year's category.

The Type 2 approach instead adds a new row each time an attribute changes, with effective-from and effective-to dates, so the fact joins to the version of the dimension that was current when the event happened. This preserves history correctly at the cost of more rows and more care in the join. Knowing when each type is appropriate — and we will work through many examples — is what keeps your historical reports honest.

```
-- Type 2: a new row per change, with validity dates and a current flag
-- dim_customers
customer_key | customer_id | city      | valid_from | valid_to   | is_current
-------------------------------------------------------------------------------
        1041 |       C-77  | Lyon      | 2024-01-01 | 2025-06-30 | false
        1042 |       C-77  | Marseille | 2025-07-01 | 9999-12-31 | true
```

Symptoms of a Bad Model

You can diagnose a struggling warehouse from its symptoms without reading a line of SQL. The same metric is defined differently in different dashboards. Nobody can state a table's grain. Every new question requires a brand-new bespoke table. Joins routinely double-count. Onboarding an analyst takes weeks because the schema has no discoverable logic and no two tables follow the same conventions.

These are not tooling problems, and no amount of compute fixes them. They are modelling problems, and they compound: each workaround makes the next one more likely. The rest of this book is a toolkit for preventing them — conformed dimensions, careful grain, disciplined handling of change — so that the warehouse stays navigable and trustworthy even as it grows to hundreds of tables and dozens of contributors.

How This Book Is Organised

We build outward from the foundations laid here. The next chapters go deep on grain and fact-table design, then on dimensions and the many flavours of slowly changing dimension, then on the harder cases — many-to-many relationships, late-arriving data, and the perennial debate about when a single

wide table beats a star schema.

Throughout, the examples are warehouse-agnostic; the patterns are identical whether your SQL runs on one cloud platform or another. What you are really learning is a way of thinking about analytical data that has survived four decades of changing technology — because it is about how humans ask questions, which has not changed at all.

This is a free sample

You've reached the end of the sample chapter.

Get the complete book — every chapter, fully worked — at **dataforgebooks.com**.

FULL EDITION · 232 PAGES · PDF

Read the full title at **dataforgebooks.com**

Questions? support@dataforgebooks.com