

## POSTS

# Database Design for Google Calendar: a tutorial

May 20, 2024

Author: Alexey Makhotkin [squadette@gmail.com](mailto:squadette@gmail.com).

## Introduction

In this database design tutorial (*~9000 words*) I'm going to show how to design the database tables for a real-world project of substantial complexity.

We'll design a clone of Google Calendar. We will model as much as possible of the functionality that is directly related to the calendar.

This series illustrates an approach explained in my "Database Design Book". Here is the website of the book: (<<https://databasedesignbook.com/>>). You can leave your email address to receive occasional updates on the book and related materials.

We will first build a complete logical model that describes the calendar data to be stored. This should take most of the effort (~80% of text by word count). After the logical model is finished, we'll build table design directly based on the logical model.

## Intended audience

The goal of the book is to help you get from a vague idea of what you need to implement (e.g.: "I need to build a website to manage schedule and instructor appointments for our gym"), to the full and complete definition of database tables.

The first three quarters of the text require only a general understanding of what databases are and how information is stored there. Parts 1-6 only talk about logical models, that do not depend on the specific database that you use (MySQL, Postgres, other classic relational servers, NoSQL solutions, cloud databases, etc.). So the majority of the text describes how the business requirements are modeled.

The last quarter of the text shows how to get from the logical model to physical tables structure. This part is absolutely not comprehensive, it only shows one of many possible approaches to the database table design. This approach, however, is perfectly valid for not very demanding systems. Also, many existing systems are designed in part using this strategy. This part of the text requires more familiarity with common databases: how the tables are created, what physical data types exist, what is primary key and index, how the tables would be queried, and how to insert and update data.

## Table of contents

- [Introduction](#)
- [Intended audience](#)

- [Approach of this book](#)
- [Problem description](#)
- [Part 1: Basic all-day events](#)
  - [Anchors](#)
  - [Attributes of User](#)
  - [Attributes of DayEvent](#)
  - [Links](#)
  - [A peek into the physical model](#)
- [Part 2: Time-based events](#)
  - [Time zones](#)
  - [Anchors](#)
  - [Attributes of Timezone](#)
  - [Attributes of TimeEvent](#)
  - [Links](#)
  - [Similarities between DateEvent and TimeEvent](#)
- [Part 3. Repeated all-day events](#)
  - [Attribute #1, cadence](#)
  - [Attribute #2, tangled attributes](#)
  - [Attribute #3](#)
  - [Days of the week: micro-anchors](#)
  - [Are we done?](#)
  - [Repeat limit: more tangled attributes](#)
- [Part 4. Rendering the calendar page](#)
  - [A note on tempo](#)
  - [General idea](#)
  - [Day slots](#)
  - [Exercise: TimeSlots](#)
  - [How far ahead do you need to think?](#)
- [Part 5. Rendering the calendar page: time-based events.](#)
- [Part 6. Complete logical model so far](#)
- [Part 7. Creating SQL tables](#)
  - [Anchors: choose names for tables](#)
  - [Attributes: choose the column name and physical type](#)
  - [1:N Links](#)
  - [M:N links](#)
  - [Finally: the tables](#)
- [Conclusion](#)
  - [What's next?](#)

## Approach of this book

Often people start with designing the tables right away, but we take a different approach. This tutorial is aimed at people who are new to database design. The goal of the process is to answer several important questions:

- Where to begin?
- How to make sure that we did not miss anything?
- how to ask for feedback on our database design;

- how to fix design mistakes;

We begin with a logical model, written in a simple tabular format. We use short formalized sentences to define data attributes and relationships between entities. This helps us to make sure that our logical model aligns with the actual business requirements. Logical model is independent from a specific database implementation.

As the second step, when the logical model is decided, we design the physical tables. This process is very straightforward. For each element of the logical model there would be a corresponding table or column. Physical models can be as dependent on a specific database implementation as you need.

## Problem description

We're going to implement a big part of Google Calendar functionality. Some parts we'll skip, but we'll try and implement every feature of calendaring. Some areas we'll implement just enough to be able to discuss the more interesting parts. In the end you will be able to add missing functionality to the schema, going through the same process.

Google Calendar is a multi-user system. For example, users can share the events with other people. We're going to implement only a bare minimum of user-related data.

Events are the central part of Google Calendar, and we're going to design them as closely as possible to the real thing. Events have title and description, as well as some other minor attributes such as location.

The most complex part of calendar events is times and dates:

- “All day” events vs time-based;
- Both can be repeated and non-repeated;
- All day events:
  - Can spread over multiple days;
- Time-based events:
  - Can have associated time zone;
  - Have begin and end time;
  - Begin and end time can happen on different days;
  - Begin and end time can be in different timezones;
- Both all-day and time based events:
  - Can be repeated daily, or every N days;
  - Can be repeated weekly, on certain days of the week; again, it can be every two or more weeks;
  - Can be repeated monthly, on a certain day or day of the week;
  - Can be repeated annually;
  - Repeating events can go on forever, until a certain date, or for a certain number of repetitions;

For the repeating events, the specific instances can be moved to a different date/time. You can also delete some instances of the repeating events, for example, skipping a certain weekly meeting.

You can change the schedule of the repeated event, even if some of the events already happened. For example, you can switch from two project meetings every week on Tuesday and Thursday to one meeting every two weeks, on Fridays.

Here is a screenshot of a day event editing form:

# Company retreat

Event

Task

Sunday, January 14, 2024 – Monday, January 15, 2024

☒

All day

Does not repeat

Does not repeat

Daily

Weekly on Sunday

Monthly on the second Sunday

Annually on January 14

Every weekday (Monday to Friday)

Custom...

More options

Save

## Part 1: Basic all-day events

### Anchors

First thing we need is to find some so-called anchors. Anchors are also known as entities. Anchors are usually nouns, such as *User* and *Event*.

Anchors are something that can be counted. “*No users*”, “*one user*”, “*two users*”, etc. Also, one defining characteristic of an anchor is that it can be added: “adds a user record to the database”.

Anchors are extremely obvious in simple cases, but may get tricky in non-obvious cases. We’re going to write down even the most obvious anchors, to get some experience with handling them.

The first two anchors that come to mind are:

Anchor	Physical table
User	
DayEvent	

Basically the only thing that anchors handle is IDs and counting. All the data is handled by attributes, discussed in the next section.

So, for example, in our database tables there would be a User with, say, ID=23, and DayEvent with ID=100, etc.

We won’t be dealing with the last column (“Physical table”) for now, we’ll discuss the physical model in the “Creating SQL Tables” section below.

To validate that we have an anchor, we can write a sentence using the name of this anchor. If this sentence makes sense, then this is an anchor. Example sentences:

- “*There are 200 Users in our database*”;
- “*When this form is submitted, a new User is added to the database*”;

Same for DayEvent:

- “*There are 3000 DayEvents in our database*”;
- “*When this button is clicked, a new DayEvent is created*”;

Such sentences will be useful later in more complicated cases.

If the sentence does not make sense, then it may be an attribute:

- “*There are 400 Prices in our database*” (???)
- “*When this form is submitted, a new Price is added to the database*” (???)

## Attributes of User

Attributes store the actual information about anchors.

Which data about users should we model? Users are ubiquitous, and different systems may want to store a lot of information about users. For this post, we’re going to model just the bare minimum of user data: emails.

Anchor	Question	Logical type	Example value	Physical column	Physical type
User	What is the email of this User?	string	“cjdate@example.org”		

What can we see here?

- This attribute belongs to the User **anchor** that was defined in the previous section;
- We use **questions** to describe all sorts of attributes. Later on we'll discuss why we prefer this style over "User's email" and such;
- **Logical type** is quite simple. If you expected to see "VARCHAR(128)" here or something like that: no, we'll discuss this much later.
- We show **an example value** that helps us to confirm our thinking. Again, in simple cases this is very obvious, but it would help reviewers to confirm that everyone is on the same page.
- We won't be dealing with two last columns for now, we'll discuss **the physical model** later in this post.

We're going to see more examples of logical types later. We extensively discuss the logical types in the book.

Let's skip ahead a little bit, and show how the table data about the users looks like. We'll be using a simple strategy to design physical tables, so the result is going to be completely unsurprising:

Table users		
id	email	...
2	"cjdate@example.org"	...
3	"someone@else.com"	...
...	...	...

This is just to show the part of the final result so that you know where we're heading to. Here, a user with ID=2 has email "cjdate@example.org", and a user with ID=3 has email "someone@else.com".

Except for this, we won't be talking much about the users here.

## Attributes of DayEvent

Suppose that we want to schedule a two-day company retreat that begins on January 14th, 2024. In terms of anchors, this is going to be a DayEvent.

Looking at the paragraph above, we can see that we need to store the following data about the DayEvents:

- Name of the event;
- Begin date and end date of the event.

Let's write that down in our table:

Anchor	Question	Logical type	Example value	Physical column	Physical type
DayEvent	What is the name of this DayEvent?	string	"Company retreat"		
DayEvent	When does the DayEvent begin?	date	2024-01-14		

Anchor	Question	Logical type	Example value	Physical column	Physical type
DayEvent	When does the DayEvent end?	date	2024-01-15		

What can we see here?

- We defined our first three attributes;
- We don't have any short names for the attributes, and this may bother us a little. We'd expect to have something like "*DayEvent\_name*" or some other identification to refer to the attribute later in the text. We'll return to this topic later.
- We have a new logical type: date. We won't need to deal with timezones in this section.
- For most events in actual calendars the begin date and end date would probably be the same (most events are single-day). We'll just store the same date in both attributes. This allows us to treat the special case (single-day events) as the general case (multi-day events). This is a general design strategy, but we're going to investigate later if this line of thinking is always applicable.

## Links

Where do we store the information that this particular user has created this particular DayEvent? At the first glance, this may look like an attribute of DayEvent, right? (Actually, no).

Anchor	Question	Logical type	Example value	Physical column	Physical type
DayEvent	Which User has created this DayEvent???	number???	2???		

Attributes cannot contain IDs. Instead, when two anchors are involved, we need to use links.

⚓ Anchor <sub>1</sub> : ⚓ Anchor <sub>2</sub>	Cardinality	Sentences (subject, verb, object, cardinality)	Physical table or column
User : DayEvent	1:N	User creates <i>several</i> DayEvents DayEvent is created by <i>only one</i> User	

Links help to pin down the correct design for many complicated scenarios. To describe the link, you need to write down two sentences. If the sentences don't make sense or do not match the reality of the business — you've just prevented a design mistake!

What do we see in this table?

- Link connects **two anchors**. Anchors could be different, like here, or the same. For example, "*Employee is a manager of Employee*" is one example of one such link.
- We use one of three **cardinalities**: **1:N**, **M:N**, and **1:1**. For cardinality **1:N** we put the anchor which is on the 1-side first. For other cardinalities, we use the order that feels better.
- We use **two formalized sentences** that involve two anchors, a verb, and information about cardinality. Those sentences allow us to validate and document our design.

(We'll discuss the links much more, of course).

## A peek into the physical model

Again skipping ahead: if we would have stopped right now, and tried to write down the physical design for the schema that we have so far, here is what we'd see. This is just to confirm that we're heading in a familiar direction.

Table: day_events				
id	name	begin_date	end_date	user_id
20	"Company retreat"	2024-01-14	2024-01-15	3
...	...	...	...	...

We'll discuss the physical model later in the "Creating SQL Tables" section.

## Part 2: Time-based events

In the previous section, we discussed basic non-repeating date-based events. Let's see how our modeling approach handles time-based events.

We modeled date-based events as the *DayEvent* anchor with the following attributes:

- *What is the name of this DayEvent?*
- *When does the DayEvent begin?*
- *When does the DayEvent end?*

Also, we defined the link between User and DayEvent: "*User creates several DayEvents*".

Let's write down a quick draft of time-based events and see how it compares to date-based events. Quoting the "problem description" section:

- "Time-based events:
  - Can have associated time zone;
  - Have begin and end time;
  - Begin and end time can happen on different days;
  - Begin and end time can be in different timezones;"

## Time zones

Every country and territory uses one or more time zones. Time zone definitions occasionally change. Each country, being a sovereign state, can decide to change their time zone definition.

Time zones may use Daylight Savings Time, or can be uniform. New time zones may be introduced, or retired. In this text, we won't go into complications of handling time zone definitions. If you were really implementing a serious global calendaring solution, you'd probably have a separate team dealing with such issues.

However, in this tutorial we will implement full timezone-aware events that are usable in practice.

We have one motivating example: **plane tickets**. Planes often cross time zone boundaries, and the take-off and landing times in your ticket would be in different time zones. Say, there is a flight from Amsterdam to London

that departs on December 24, at 16:50 (Amsterdam time) and lands at 17:05 (London time). So, the flight duration is 1 hour 15 minutes.

Time zones inspire a lot of programming folklore. There are many blog posts, horror stories, “things every programmer should know” and other texts related to time zones, particularly in database context. Also, many systems keep breaking in various ways around the daylight switch time. We will discuss only as much as needed for our purposes, and briefly mention some important things to consider.

## Anchors

Having said that, it seems that we need to add two anchors:

Anchor	Physical table
Timezone	
TimeEvent	

There are dozens of time zones in the world. We can confirm the validity of the Timezone anchor by writing down example sentences:

- *“There are 120 Timezones in our database”;*
- *“When this import script finishes, a new Timezone is added to our database”.*

(Timezone data structure is discussed below.)

The sentences for TimeEvent are also straightforward:

- *“There are 2500 TimeEvents in our database”;*
- *“When this button is clicked, a new TimeEvent is created”;*

## Attributes of Timezone

For the purposes of this text, we’ll do only a very minimal model of Timezone. Basically, the only attribute we’d introduce is:

Anchor	Question	Logical type	Example value	Physical column	Physical type
Timezone	What is the human-readable name of this Timezone?	string	<i>“Europe/Kyiv”</i>		

We won’t go into details of how the timezone is actually defined. We assume that there is a complementary logical model that describes the structure of timezones. Also, we assume that there is some function that takes a local time in the specified timezone and returns UTC time (and vice-versa). This will be discussed in more detail in the next section, when we will talk about repeating events.

For clarity, here is what else would be included into a time zone definition:

- What is the UTC offset of this time zone?
- Does this time zone have Daylight Savings Time?
- When does DST begin? When does DST end?
- What is the UTC offset when DST is on?

- We'd also need to model previous definitions of a time zone. For example, the government may decide to change the day when DST goes into force, or get rid of DST, etc.
- Is this time zone active or retired?

This is an incomplete list. Modeling all of this data using our approach is possible, but is a separate, and quite technical exercise.

Let's get back to events.

## Attributes of TimeEvent

Anchor	Question	Logical type	Example value	Physical column	Physical type
TimeEvent	What is the name of this TimeEvent?	string	"Catch-up meeting"		
TimeEvent	When does the TimeEvent begin?	date/time (local)	2024-01-14 12:30		
TimeEvent	When does the TimeEvent end?	date/time (local)	2024-01-14 13:15		

Note that we're using local time here. You may have read that time should be stored in UTC time (without any time zones), and then formatted for human readability using a preferred time zone.

Here we have a different situation. Time zones can change. Suppose that we scheduled a billiard game on September 6, 2058, from 09:30 to 11:00, Cologne time. At the moment we don't know what UTC offset is going to be in that time zone at that time. So we must store the data exactly as entered by the user, and then adjust it as the local legislation changes.

## Links

We have two very similar links here.

⚓ Anchor <sub>1</sub> : ⚓ Anchor <sub>2</sub>	Cardinality	Sentences (subject, verb, object, cardinality)	Physical table or column
Timezone : TimeEvent	1:N	Timezone is used for the <b>start</b> time of <i>several</i> TimeEvents  TimeEvent uses <i>only one</i> Timezone for the <b>start</b> time	
Timezone : TimeEvent	1:N	Timezone is used for the <b>end</b> time of <i>several</i> TimeEvents  TimeEvent uses <i>only one</i> Timezone for the <b>end</b> time	
User : TimeEvent	1:N	User creates <i>several</i> TimeEvents	

⚓ <b>Anchor<sub>1</sub> :</b> ⚓ <b>Anchor<sub>2</sub></b>	<b>Cardi- nality</b>	<b>Sentences (subject, verb, object, cardinality)</b>	<b>Physical table or column</b>
		TimeEvent is created by <i>only one</i> User	

The definitions of both links differ only in a single word (“start” vs “end”).

Most of the time-based events would have the same time zone for both start and end times. We design this as a general case: we always specify both, even if they are the same. This approach would help us to get used to handling more complicated cases.

## Similarities between DateEvent and TimeEvent

All-day events and time-based events look similar. Does it make sense to think about unification?

For example, both events have names. Also, how different are “date” and “date with time” from each other? We can also observe that both types of events would have more common data, such as “location”, list of invited guests, “description”, etc. Maybe we could extract that to some component shared between those two anchors?

Thankfully, logical modeling allows us to wait a little bit before making that decision. After all, it’s just simple tables that we can reshuffle before we commit to a physical table implementation.

For now, though, we want to gather more information on similarities (and, more important, dissimilarities) of two types of events that we’ve seen so far. Also, we want to see if there would be more types of events, and what attributes and links they have.

## Part 3. Repeated all-day events

As we may remember from the initial problem description:

*“Both all-day and time based events:*

- *Can be repeated daily, or every N days;*
- *Can be repeated weekly, on certain days of the week; again, it can be every two or more weeks;*
- *Can be repeated monthly, on a certain day or day of the week;*
- *Can be repeated annually;*
- *Repeating events can go on either forever, or until a certain date, or for a certain number of repetitions;”*

In this section we’ll only talk about all-day events. Later we’ll see how the Minimal Modeling method handles commonalities between different anchors, in this case time-based events. Also, we’ll see how the logical schema gets changed: we would use this as an example of how the draft design is edited when a better design approach is introduced. Remember that all of this happens before we even start thinking about database tables, so this is a very smooth process: you don’t need to worry about table migrations yet.

If you think about the bullet points listed above, your reaction may be: “we probably need JSON for that”. That may well be true, but JSON belongs to the physical table design, so we won’t discuss this at this point. We’ll design everything that is needed on a logical level, and later on we’ll see what physical options we have.

### Attribute #1, cadence

So, let’s ask the first question, hoping that it would help us find an attribute: “*How often is that event repeated?*” Looking at event editing form we can see possible answers to that question: a) never; b) daily; c)

weekly; d) monthly; e) annually.

We say that such attributes have an “either/or/or” type. Let’s write it down as an attribute.

Anchor	Question	Logical type	Example value	Physical column	Physical type
DayEvent	How often is that DayEvent repeated?	either/or/or	<i>daily weekly monthly annually</i>		

For either/or/or attributes, we show the entire list of possible values in the “Example value” column.

What do we do with never-repeating events? On the logical level, **an attribute can either be set to a specific value, or unset: this is a basic principle of minimal modeling**. So if this attribute is not set then the event would be non-repeating.

## Attribute #2, tangled attributes

Reading further, we see that for all four frequencies, there is an additional possibility. The events can repeat every N days, N weeks, N months, and N years. Let’s write that down.

Anchor	Question	Logical type	Example value	Physical column	Physical type
DayEvent	For repeated events: what is the repetition step?	integer	<i>2 (every two days/weeks/etc)</i>		

This is our first example of **tangled attributes**. Its value only makes sense when another attribute is set. We specify that by adding “*For repeated events:*” as part of the question.

Note that this is only a human-readable notation, we’re not going to discuss here how to write a machine-readable logical schema.

## Attribute #3

When you specify a monthly event, you have two options: repeat on the same day of the month (say, every 16th of the month), or repeat on the same weekday of the month as the original date (say, every second Tuesday). The base date is taken from the normal “When does the DayEvent begin?” attribute that was discussed in the beginning.

Let’s try and define this attribute:

Anchor	Question	Logical type	Example value	Physical column	Physical type
DayEvent	For monthly repeated events: which day of the month does it fall on?	either/or/or	<i>same_day same_weekday</i>		

## Days of the week: micro-anchors

Events can be repeated weekly, on certain days of the week. So, for example we can have a weekly event that happens on Mondays, Wednesdays and Fridays. Where do we store this?

Let's start with an attribute that may look like this:

Anchor	Question	Logical type	Example value	Physical column	Physical type
<b>DayEvent</b>	<b>For weekly repeated events: which days of the week does it fall on?</b>	<b>Array of strings???</b>	<i>["Mon", "Wed", "Fri"] ???</i>		

Would it work? Modern database systems such as Postgres and MySQL natively support storing arrays: Postgres has an array type, also they both have JSON type, so you can store an entire array in a single table column. Even though we discuss the logical level, this parallel can help to at least confirm the plausibility of this approach. Also, it's entirely possible that we'll decide to store that information in exactly this way, when we will discuss the physical table schema.

However, the approach taken by Minimal Modeling demands that we introduce a new anchor called *DayOfTheWeek*.

Anchor	Physical table	ID example
<b>DayOfTheWeek</b>		"Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"

This is an example of a well-known anchor, similar to currencies, languages and countries. We discuss well-known anchors in <https://minimalmodeling.substack.com/p/currencies-countries-and-languages>.

*DayOfTheWeek* may look a bit awkward because it is so tiny. There are only seven possible IDs here, and new ones will never be added. Also, the IDs are unusual because they are strings, and not the commonly used integer numbers. But introducing it helps keep the modeling approach simple.

You can also introduce attributes for this anchor, such as human-readable names of the days of the week. This is left as an exercise for the reader.

Now back to our task. We have two anchors: *DayEvent* and *DayOfTheWeek*. To connect them, we need a link.

$\clubsuit$ <b>Anchor<sub>1</sub> : <math>\clubsuit</math> Anchor<sub>2</sub></b>	<b>Cardinality</b>	<b>Sentences (subject, verb, object, cardinality)</b>	<b>Physical table or column</b>
<b>DayEvent : DayOfTheWeek</b>	<b>M:N</b>	<p>For weekly repeated DayEvents:</p> <p>DayEvent may happen on <i>several</i> DaysOfTheWeek</p> <p>DayOfTheWeek can contain <i>several</i> DayEvents</p>	

This is a pretty normal link, but tangled, like the attributes we've seen above. This link only makes sense when it corresponds to the DayEvent that is repeated weekly.

**Are we done?**

How do we know when we’re done with modeling? This is a very common concern among beginners (as well as experienced database designers). Our approach helps us **self-check our logical model against our business requirements**.

Let’s revisit the original requirements again, and highlight the parts that we’ve covered so far:

- “Both **all-day** and time based events:
- Can be repeated **daily**, or **every N days**;
  - Can be repeated **weekly**, on **certain days of the week**; again, it can be **every two or more weeks**;
  - Can be repeated **monthly**, on a **certain day** or **day of the week**;
  - Can be repeated **annually**;
  - Repeating events can go on forever, until a certain date, or for a certain number of repetitions;"

Okay, we can now see that we forgot about the number of event repetitions. The corresponding part of requirements is not marked in any way. Let’s fix this.

Repeat limit: more tangled attributes

“For how long do the periodic events repeat?” This looks like a plausible sentence to define the attribute. There are three possible answers: “forever”, “until a certain date”, and “for a certain number of reps”.

Anchor	Question	Logical type	Example value	Physical column	Physical type
DayEvent	For repeated events: for how long does the DayEvent repeat?	either/ or/or	forever until_date N_repetitions		

Now we can add two final missing pieces: which date, and how many repetitions.

Anchor	Question	Logical type	Example value	Physical column	Physical type
DayEvent	For events repeated until a certain date: what is the date?	date	2024-01-17		
DayEvent	For events repeated for a certain number of reps: how many reps?	integer	10		

We’re done.

Let’s quickly summarize the pieces of data that we defined here:

- anchor: *DayEvent*
  - attribute: How often is it repeated?
    - attribute: (for monthly) Which day does it fall on?
    - link: (for weekly) falls on certain *DaysOfTheWeek*;
  - attribute: What is the repetition step?
  - attribute: For how long is it repeated?

- attribute: (until a certain date) When does the repetition end?
- attribute: (for number of reps) How many times is it repeated?
- anchor: *DayOfTheWeek*

One extra anchor; six attributes, some of them tangled; one link.

## Part 4. Rendering the calendar page

So far we've discussed the book-keeping part of the calendar. We have only one record for the entire series of events. Ten weekly project status meetings correspond to a *single* database record. Unlimited number of birthdays of our friend correspond to a *single* database record.

Let's get back to the application that we're working on: a calendar. We need to show a weekly view of the user's calendar: say, seven days starting from 26th of Feb up to 3rd of Mar. Which events do we need to show on that page? Say, there is one of the weekly project status meetings (out of ten) happening that week. If the birthday (annual event) falls on that week we need to show it.

So, we need to write some sort of SQL query that looks like this pseudo-code:

```
SELECT ...
FROM ...
WHERE <date> BETWEEN '2024-02-26' AND '2024-03-26';
```

Maybe this would even be several SQL queries, or even some code in a programming language. The data structure that we've considered so far is quite complicated. To find the events that must be shown on a certain week, you need to take a lot into account. This may quickly become impractical.

### A note on tempo

I've been writing this chapter in the course of a few months. I've been thinking about this problem a lot. I have a quite clear understanding of the end state that I have in mind, I just need to write it down.

But if I were to present to you the complete table design right now, it wouldn't be useful for our goal: to learn database design. You won't understand why I made certain decisions.

At the same time, I don't want to present very small incremental changes, so that text is not too long. So we need to find some middle ground.

We started this section with a question of how to render a weekly page. Let's remember another requirement that our calendar application definitely has: **changing and canceling some events from the series**. Say, you have ten weekly project meetings, but you want to cancel one of them because the weather is too good to miss. The existing book-keeping part of our database model won't change. But we need to add some anchors, attributes and links for the second half: rendering and modification.

### General idea

We want to introduce a new anchor that would store the information about *each event in the series*. So, if we have 10 weekly project status meetings, we're going to have ten rows in some table. Each record would correspond to a specific date (e.g., *2024-02-12*, *2024-02-19*, etc.).

First, this would make our rendering very simple. You have a very easy way to find all the events that fall on a specific day.

Second, this would allow us to reschedule and cancel some events in the series. If we have a project meeting at 12:00 normally, but on a certain week we want to move it to 14:00 (or even to a different day), we can do that. Data in the original book-keeping anchor, `TimeEvent`, that we defined previously, won't change.

Also, if we just want to skip one project status meeting, we can mark this particular day as skipped.

## Day slots

First, we have to find a name for those things. In some cases this may be a challenge. Just five minutes ago, having another cup of tea, I realized that a good word for this thing is “slot”.

Also, like before, we're going to treat per-day and time-based events differently. So, we're going to have *DaySlot* and *TimeSlot* anchors. Let's discuss per-day slots first.

Anchor	Physical table
DaySlot	

*DaySlot* needs a surprisingly small number of attributes:

Anchor	Question	Logical type	Example value	Physical column	Physical type
DaySlot	On which day does this DaySlot happen?	date	2024-02-12		
DaySlot	Is this DaySlot skipped?	yes/no	yes		

Note that the user can change the date for a specific slot! So, we see that this requirement is handled cleanly.

Also, we need to establish a link between *DaySlot* and the corresponding *DayEvent*.

⚓ Anchor <sub>1</sub> : ⚓ Anchor <sub>2</sub>	Cardinality	Sentences (subject, verb, object, cardinality)	Physical table or column
DayEvent : DaySlot	1:N	DayEvent may generate <i>several</i> DaySlots DaySlot corresponds to <i>only one</i> DayEvent	

Some things to note: first, **we always create a DaySlot for every DayEvent, even for non-repeating ones**. This is needed to simplify the rendering code.

Second, we have an interesting **problem with “infinite” events**. Suppose that we added our friend's birthday to the calendar, repeating annually. How many corresponding DaySlots do we need to create? One possible solution would be to choose some arbitrary limit such as 100 years in the future, and create all the slots for that. Other solutions are possible, such as on-demand creation when user requests to show a calendar page in some distant future.

Third, if you think about this, it's possible that more information could also be **different for each slot**. For example, it's possible that some meetings will take place in different locations. Also, the guest list may change:

you can invite extra people to a certain meeting. Also, the attendance would certainly be different. We won't cover those aspects here, because modeling this it's pretty straightforward: just add more links that connect *DaySlot* with other anchors.

Also, one thing to consider is that **date arithmetics needs a bit of care**. How do we deal with birthdays of people who were born on February 29? We will have to decide something. Maybe we'll prohibit the user from creating such events? Maybe we'll ask them where to move the slot: day earlier or day later? Similar problem also exists for monthly events that happen on the 31st day of the month.

## Exercise: TimeSlots

Here is an exercise for the determined reader. Think about how you would model the *TimeSlot* anchor, its attributes and links. Fill in the tables with anchors, links and attributes, using the format explained above. Think about what role would time zones play.

## How far ahead do you need to think?

Sometimes you can create a better design if you consider a slightly broader set of requirements. We did it in this chapter: we started thinking about rendering the page, but then we also considered the need to modify some events in the series.

Sometimes you can create a better design if you consider requirements independently. In the previous chapters, we looked at time-based and per-day events, and decided to handle them separately for now.

So far we did not even mention any hypothetical future requirements. We've only been designing stuff that we know is needed. Can we think up something that would be nice to have in the future, and design ahead? If you only include known requirements, there is a chance of overfitting design for known data points.

At the same time, people sometimes introduce considerations that never actually materialize. In this case design may introduce extra complexity that is not needed for the actual requirements, adding a bit of friction.

There is always a chance of leaning towards one of those traps, or even falling into them.

Logical design based on Minimal Modeling insists on modeling only the parts that we actually know are needed. We can afford that because Minimal Modeling treats changing requirements as a given. Also, Minimal Modeling discourages you from adding more abstract concepts, and this is intentional.

We'll discuss this aspect of physical design later in the book. We'll introduce the concept of Game of Tables, and the idea of Date's Demon.

## Part 5. Rendering the calendar page: time-based events.

For repeated time-based events, we choose the same approach as with all-day events. We're going to introduce an anchor called "*TimeSlot*". *TimeSlot* corresponds to a specific event on a specific date and time. A repeated event corresponds to several *TimeSlots*.

Time slots can be rescheduled or canceled manually, just like all-day slots.

Here is an anchor:

<b>Anchor</b>	<b>Physical table</b>
<b>TimeSlot</b>	

And the attributes:

<b>Anchor</b>	<b>Question</b>	<b>Logical type</b>	<b>Example value</b>	<b>Physical column</b>	<b>Physical type</b>
<b>TimeSlot</b>	<b>When does the TimeSlot begin?</b>	<b>date/time (local)</b>	<i>2024-01-14 12:30</i>		
<b>TimeSlot</b>	<b>When does the TimeSlot end?</b>	<b>date/time (local)</b>	<i>2024-01-14 13:15</i>		
<b>TimeSlot</b>	<b>Is this TimeSlot skipped?</b>	<b>yes/no</b>	<i>yes</i>		

A specific time slot can generally be moved even to a different day, so we have to keep this information also. Which time zone shall we use for the begin/end time? As you may remember from Part 2, in Google Calendar you can have different time zones for begin and end time. If you think about that, it makes sense to keep it for the time slots also.

⚓ <b>Anchor<sub>1</sub> :</b> ⚓ <b>Anchor<sub>2</sub></b>	<b>Cardinality</b>	<b>Sentences (subject, verb, object, cardinality)</b>	<b>Physical table or column</b>
<b>Timezone :</b> <b>TimeSlot</b>	<b>1:N</b>	Timezone is used for the <b>start</b> time of <i>many</i> TimeSlots  TimeSlot uses <i>only one</i> Timezone for the <b>start</b> time	
<b>Timezone :</b> <b>TimeSlot</b>	<b>1:N</b>	Timezone is used for the <b>end</b> time of <i>many</i> TimeEvents  TimeSlot uses <i>only one</i> Timezone for the <b>end</b> time	

Also, we need to connect TimeSlots with TimeEvents, same as we did with DaySlots/DayEvents:

⚓ <b>Anchor<sub>1</sub> :</b> ⚓ <b>Anchor<sub>2</sub></b>	<b>Cardinality</b>	<b>Sentences (subject, verb, object, cardinality)</b>	<b>Physical table or column</b>
<b>TimeEvent :</b> <b>TimeSlot</b>	<b>1:N</b>	TimeEvent may generate <i>several</i> TimeSlots  TimeSlot corresponds to <i>only one</i> TimeEvent	

Same as with *DaySlot*, we would create a *TimeSlot* even for non-repeated TimeEvents.

## Part 6. Complete logical model so far

Let's go back and collect everything that we've designed so far.

First, the complete list of anchors (7 in total):

Anchor	Physical table	ID example
User		
Timezone		
DayEvent		
TimeEvent		
DayOfTheWeek		"Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"
DaySlot		
TimeSlot		

Second, list of attributes (ordered by anchor):

Anchor	Question	Logical type	Example value	Physical column	Physical type
User	What is the email of this User?	string	<i>"cjdate@example.org"</i>		
Timezone	What is the human-readable name of this Timezone?	string	<i>"Europe/Kyiv"</i>		
DayEvent	What is the name of this DayEvent?	string	<i>"Company retreat"</i>		
DayEvent	When does the DayEvent begin?	date	<i>2024-01-14</i>		
DayEvent	When does the DayEvent end?	date	<i>2024-01-15</i>		
DayEvent	How often is that DayEvent repeated?	either/or/or	<i>daily weekly monthly annually</i>		
DayEvent	For repeated events: what is the repetition step?	integer	<i>2 (every two days/weeks/etc)</i>		
DayEvent	For monthly repeated events: which day of the month does it fall on?	either/or/or	<i>same_day same_weekday</i>		
DayEvent	For repeated events: for how long does the DayEvent repeat?	either/or/or	<i>forever until_date N_repetitions</i>		

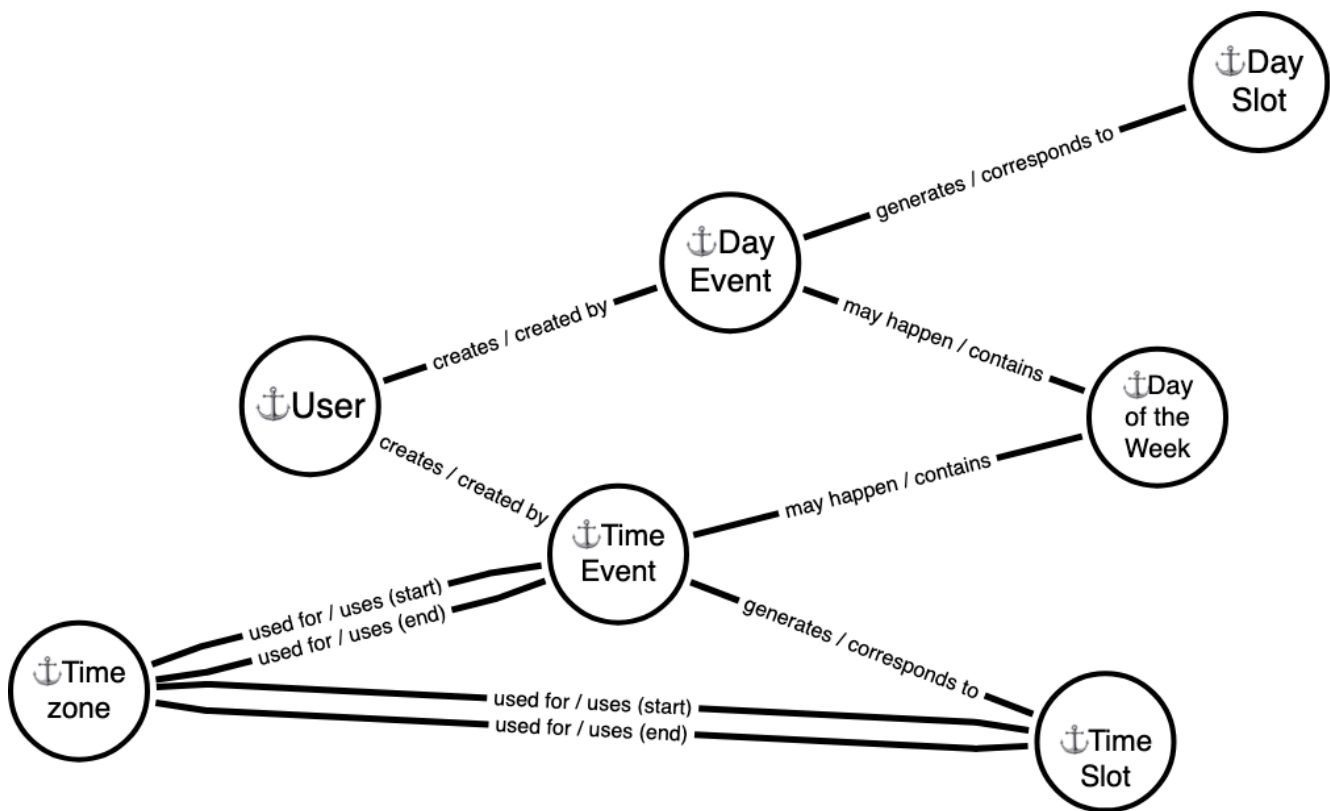
Anchor	Question	Logical type	Example value	Physical column	Physical type
DayEvent	For events repeated until a certain date: what is the date?	date	2024-01-17		
DayEvent	For events repeated for a certain number of reps: how many reps?	integer	10		
TimeEvent	What is the name of this TimeEvent?	string	"Catch-up meeting"		
TimeEvent	When does the TimeEvent begin?	date/time (local)	2024-01-14 12:30		
TimeEvent	When does the TimeEvent end?	date/time (local)	2024-01-14 13:15		
DaySlot	On which day does this DaySlot happen?	date	2024-02-12		
DaySlot	Is this DaySlot skipped?	yes/no	yes		
TimeSlot	When does the TimeSlot begin?	date/time (local)	2024-01-14 12:30		
TimeSlot	When does the TimeSlot end?	date/time (local)	2024-01-14 13:15		
TimeSlot	Is this TimeSlot skipped?	yes/no	no		

Third, list of links (in no particular order):

$\hookrightarrow$ Anchor <sub>1</sub> : $\hookrightarrow$ Anchor <sub>2</sub>	Cardinality	Sentences (subject, verb, object, cardinality)	Physical table or column
User : DayEvent	1:N	User creates <i>many</i> DayEvents DayEvent is created by <i>only one</i> User	
User : TimeEvent	1:N	User creates <i>many</i> TimeEvents TimeEvent is created by <i>only one</i> User	
Timezone : TimeEvent	1:N	Timezone is used for the start time of <i>many</i> TimeEvents TimeEvent uses <i>only one</i> Timezone for the start time	

$\hookrightarrow$ Anchor <sub>1</sub> : $\hookrightarrow$ Anchor <sub>2</sub>	Cardinality	Sentences (subject, verb, object, cardinality)	Physical table or column
<b>Timezone : TimeEvent</b>	<b>1:N</b>	Timezone is used for the end time of <i>many</i> TimeEvents  TimeEvent uses <i>only one</i> Timezone for the end time	
<b>DayEvent : DayOfTheWeek</b>	<b>M:N</b>	For weekly repeated DayEvents:  DayEvent may happen on <i>several</i> DaysOfTheWeek  DayOfTheWeek can contain <i>several</i> DayEvents	
<b>TimeEvent : DayOfTheWeek</b>	<b>M:N</b>	For weekly repeated TimeEvents:  TimeEvent may happen on <i>several</i> DaysOfTheWeek  DayOfTheWeek can contain <i>several</i> TimeEvents	
<b>DayEvent : DaySlot</b>	<b>1:N</b>	DayEvent may generate <i>several</i> DaySlots  DaySlot corresponds to <i>only one</i> DayEvent	
<b>TimeEvent : TimeSlot</b>	<b>1:N</b>	TimeEvent may generate <i>several</i> TimeSlots  TimeSlot corresponds to <i>only one</i> TimeEvent	
<b>Timezone : TimeSlot</b>	<b>1:N</b>	Timezone is used for the start time of <i>many</i> TimeSlots  TimeSlot uses <i>only one</i> Timezone for the start time	
<b>Timezone : TimeSlot</b>	<b>1:N</b>	Timezone is used for the end time of <i>many</i> TimeEvents  TimeSlot uses <i>only one</i> Timezone for the end time	

Finally, here is a diagram that shows all anchors and links (but not attributes):



## Part 7. Creating SQL tables

In the previous chapters we defined the complete logical model, so most of the work is actually already done. The rest is pretty straightforward.

For teaching purposes we're going to use one specific table design strategy: "one table per anchor". It is one of the most common approaches to physical table design. There are several more possible strategies, we're going to discuss them in a book.

We have 7 anchors, 21 attributes and 10 links so far. Given that we use "one table per anchor" strategy, we're going to have  $7 + 2 = 9$  tables (number of anchors + number of M:N links), and  $21 + 8 = 29$  columns in total (number of attributes + number of 1:N links).

If our logical design correctly describes the business requirements then the tables will be automatically correct. We'll talk about evolving requirements in the book. Also, we'll discuss design mistakes and how to fix them.

We're going to revisit the tables from the previous section, and fill in our choices:

- For anchors, fill in "Physical table" columns;
- For each attribute, fill in "Physical column", and choose the "Physical type";
- For each M:N link, choose the name of the physical table;
- For each 1:N link, fill in the column name in the table that corresponds to the N-side anchor;

### Anchors: choose names for tables

Here we just choose a straightforward plural name for each table.

Anchor	Physical table	ID example
User	users	
Timezone	timezones	
DayEvent	day_events	
TimeEvent	time_events	
DayOfTheWeek	days_of_the_week NB: <i>this table may be virtual, see below</i>	“Mon”, “Tue”, “Wed”, “Thu”, “Fri”, “Sat”, “Sun”
DaySlot	day_slots	
TimeSlot	time_slots	

Some companies or applications enforce different naming conventions (singular, camel case, etc.). In that case, you would just use the names that comply with the convention.

## Attributes: choose the column name and physical type

For the **physical column name**, we choose a sensible name. For example:

- `day_events.end_date` would be the column name for the “When does the DayEvent begin?” attribute;
- `time_slots.is_skipped` would be the column name for the “Is this TimeSlot skipped?” attribute

And so on. Due to the way relational databases work, you have to choose a very short name. In many cases this name by itself is not enough to fully explain the meaning of the data. That’s one of the reasons why we begin with the logical schema, and use longer human-readable questions to define the semantics of the attributes.

For the **physical type**, we choose a sensible type without much discussion. This topic is covered extensively in the book. There is also a list of recommended data types for each logical type in the book, and we just use that directly.

If you’re working with an existing system, you may be required to choose an alternative physical data type for the column. For example, your database server may support a better suited data type; or there could be some engineering guidelines that make you choose a different data type.

We’re going to discuss this variance in the book. However, full discussion of all physical design concerns is well outside of the scope of *any* book. This is the stuff that you spend your career on learning, and it changes as new technologies emerge.

Anchor	Question	Logical type	Example value	Physical column	Phy
User	What is the email of this User?	string	“cjdate@example.org”	users.email	VAR NOT
Timezone	What is the human-readable name of	string	“Europe/Kyiv”	timezones.name	VAR NOT

Anchor	Question	Logical type	Example value	Physical column	Phy
	this Timezone?				
DayEvent	What is the name of this DayEvent?	string	<i>“Company retreat”</i>	day_events.name	VAR NOT
DayEvent	When does the DayEvent begin?	date	<i>2024-01-14</i>	day_events.begin_date	DAT NUL
DayEvent	When does the DayEvent end?	date	<i>2024-01-15</i>	day_events.end_date	DAT NUL
DayEvent	How often is that DayEvent repeated?	either/or/or	<i>daily weekly monthly annually</i>	day_events.repeated	VAR NUL
DayEvent	For repeated events: what is the repetition step?	integer	<i>2 (every two days/weeks/etc)</i>	day_events.repetition_step	INT
DayEvent	For monthly repeated events: which day of the month does it fall on?	either/or/or	<i>same_day same_weekday</i>	day_events.repeated_monthly_on	VAR NUL
DayEvent	For repeated events: for how long does the DayEvent repeat?	either/or/or	<i>forever until_date N_repetitions</i>	day_events.repeated_until	VAR NUL
DayEvent	For events repeated until a	date	<i>2024-01-17</i>	day_events.repeated_until_date	DAT

Anchor	Question	Logical type	Example value	Physical column	Phy
	<b>certain date: what is the date?</b>				
<b>DayEvent</b>	<b>For events repeated for a certain number of reps: how many reps?</b>	<b>integer</b>	<i>10</i>	<code>day_events.repeated_reps</code>	INT
<b>TimeEvent</b>	<b>What is the name of this TimeEvent?</b>	<b>string</b>	<i>“Catch-up meeting”</i>	<code>time_events.name</code>	VAR NOT
<b>TimeEvent</b>	<b>When does the TimeEvent begin?</b>	<b>date/time (local)</b>	<i>2024-01-14 12:30</i>	<code>time_events.begin_local_time</code>	DAT NUL
<b>TimeEvent</b>	<b>When does the TimeEvent end?</b>	<b>date/time (local)</b>	<i>2024-01-14 13:15</i>	<code>time_events.end_local_time</code>	DAT NUL
<b>DaySlot</b>	<b>On which day does this DaySlot happen?</b>	<b>date</b>	<i>2024-02-12</i>	<code>day_slots.the_date</code>	DAT NUL
<b>DaySlot</b>	<b>Is this DaySlot skipped?</b>	<b>yes/no</b>	<i>yes</i>	<code>day_slots.is_skipped</code>	TIN UNS NUL 0
<b>TimeSlot</b>	<b>When does the TimeSlot begin, in local time?</b>	<b>date/time (local)</b>	<i>2024-01-14 12:30</i>	<code>time_slots.begin_local_time</code>	DAT NUL
<b>TimeSlot</b>	<b>When does the TimeSlot end, in local time?</b>	<b>date/time (local)</b>	<i>2024-01-14 13:15</i>	<code>time_slots.end_local_time</code>	DAT NUL

Anchor	Question	Logical type	Example value	Physical column	Phy
TimeSlot	Is this TimeSlot skipped?	yes/no	yes	time_slots.is_skipped	TIN UNS NUL 0

For this problem, we’ve used around half of logical attribute types:

- string: 4 attributes;
- date: 4 attributes;
- either/or/or: 3 attributes;
- integer: 2 attributes;
- date/time (local): 4 attributes;
- yes/no: 2 attributes.

You can see that the physical definitions of attributes of the same logical type are almost the same. The only differences are: a) maximum length of strings; and b) NULL vs NOT NULL.

We choose “NOT NULL” for attributes where the value always needs to be there due to business requirements. For example, the name of the event, or the start date of the all-day event. For tangled attributes, we choose nullable physical types (“NULL”). We discuss nullability in the book, but note that “NULL” only exists in the physical schema.

Just as NULLs, so-called “sentinel values” also do not exist in logical modeling.

## 1:N Links

For 1:N links, we add a column to the N-side anchor table. For example:

⚓ Anchor <sub>1</sub> : ⚓ Anchor <sub>2</sub>	Cardi- nality	Sentences (subject, verb, object, cardinality)	Physical table or column
User : DayEvent	1:N	User creates <i>many</i> DayEvents DayEvent is created by <i>only one</i> User	day_events.user_id
DayEvent : DaySlot	1:N	DayEvent may generate <i>several</i> DaySlots DaySlot corresponds to <i>only one</i> DayEvent	day_slots.day_event_id

Choosing the column name is usually quite easy. The only complication could be when there are two and more different links between the same two anchors. We have that situation with the timezones, and we’ll use two different columns.

## M:N links

For M:N links we must use a separate table for each link. Every such table will have almost identical structure, only the column names would be different.

We only need to find a good name for such a table. There is no naming method that works in all cases, you will have to try some combinations, looking for readability. For links this is especially difficult because it’s not clear

which of the two anchors is more important and should come first.

Same as with attributes, due to the way relational databases work, the name of the table needs to be quite short. In many cases the name by itself is not enough to fully explain the meaning of the data. That's one of the reasons why we prepare logical schema, and use human-readable sentences to define the semantics of the links.

Anyway, here is the full table of links with the names chosen for the tables and columns (see the last column).

⚓ <b>Anchor<sub>1</sub></b> : ⚓ <b>Anchor<sub>2</sub></b>	<b>Cardi- nality</b>	<b>Sentences (subject, verb, object, cardinality)</b>	<b>Physical table or column</b>
<b>User : DayEvent</b>	<b>1:N</b>	User creates <i>many</i> DayEvents DayEvent is created by <i>only one</i> User	day_events.user_id
<b>User : TimeEvent</b>	<b>1:N</b>	User creates <i>many</i> TimeEvents TimeEvent is created by <i>only one</i> User	time_events.user_id
<b>Timezone : TimeEvent</b>	<b>1:N</b>	Timezone is used for the start time of <i>many</i> TimeEvents TimeEvent uses <i>only one</i> Timezone for the start time	time_events. start_timezone_id
<b>Timezone : TimeEvent</b>	<b>1:N</b>	Timezone is used for the end time of <i>many</i> TimeEvents TimeEvent uses <i>only one</i> Timezone for the end time	time_events. end_timezone_id
<b>DayEvent : DayOfTheWeek</b>	<b>M:N</b>	For weekly repeated DayEvents:  DayEvent may happen on <i>several</i> DaysOfTheWeek  DayOfTheWeek can contain <i>several</i> DayEvents	day_event_dows
<b>TimeEvent : DayOfTheWeek</b>	<b>M:N</b>	For weekly repeated TimeEvents:  TimeEvent may happen on <i>several</i> DaysOfTheWeek  DayOfTheWeek can contain <i>several</i> TimeEvents	time_event_dows
<b>DayEvent : DaySlot</b>	<b>1:N</b>	DayEvent may generate <i>several</i> DaySlots  DaySlot corresponds to <i>only one</i> DayEvent	day_slots. day_event_id

⌚ Anchor <sub>1</sub> : ⌚ Anchor <sub>2</sub>	Cardinality	Sentences (subject, verb, object, cardinality)	Physical table or column
<b>TimeEvent : TimeSlot</b>	<b>1:N</b>	TimeEvent may generate <i>several</i> TimeSlots  TimeSlot corresponds to <i>only one</i> TimeEvent	time_slots. time_event_id
<b>Timezone : TimeSlot</b>	<b>1:N</b>	Timezone is used for the start time of <i>many</i> TimeSlots  TimeSlot uses <i>only one</i> Timezone for the start time	time_slots. start_timezone_id
<b>Timezone : TimeSlot</b>	<b>1:N</b>	Timezone is used for the end time of <i>many</i> TimeEvents  TimeSlot uses <i>only one</i> Timezone for the end time	time_slots. end_timezone_id

## Finally: the tables

As we mentioned in the previous section, we're going to have 8 (eight) SQL tables: 6 for anchors and 2 for M:N links. One anchor (*DayOfTheWeek*) is special, so we don't create a physical table for that. We use a very common approach to designing physical tables. Other approaches are also possible, but this discussion is outside the scope of this post. So, let's just write down all the tables, and add all the attributes that we have.

This is a very straightforward and even boring process at this point.

```
CREATE TABLE users (
  id INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT,
  email VARCHAR(64) NOT NULL
);
```

```
CREATE TABLE timezones (
  id INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(64) NOT NULL
);
```

```
CREATE TABLE day_events (
  id INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT,
  user_id INTEGER NOT NULL,
  name VARCHAR(128) NOT NULL,
  begin_date DATE NOT NULL,
  end_date DATE NOT NULL,
  repeated VARCHAR(24) NULL,
  repetition_step INTEGER NULL,
  repeated_monthly_on VARCHAR(24) NULL,
  repeated_until VARCHAR(24) NULL,
  repeated_until_date VARCHAR(24) NULL,
```

```

    repeated_reps INTEGER NULL
);

CREATE TABLE time_events (
    id INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT,
    user_id INTEGER NOT NULL,
    start_timezone_id INTEGER NOT NULL,
    end_timezone_id INTEGER NOT NULL,
    name VARCHAR(128) NOT NULL,
    begin_local_time DATETIME NOT NULL,
    end_local_time DATETIME NOT NULL
);

CREATE TABLE day_slots (
    id INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT,
    day_event_id INTEGER NOT NULL,
    the_date DATE NOT NULL,
    is_skipped TINYINT UNSIGNED NOT NULL DEFAULT 0
);

CREATE TABLE time_slots (
    id INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT,
    time_event_id INTEGER NOT NULL,
    begin_local_time DATETIME NOT NULL,
    end_local_time DATETIME NOT NULL,
    start_timezone_id INTEGER NOT NULL,
    end_timezone_id INTEGER NOT NULL,
    is_skipped TINYINT UNSIGNED NOT NULL DEFAULT 0
);

CREATE TABLE day_event_dows (
    day_event_id INTEGER NOT NULL,
    day_of_week VARCHAR(3) NOT NULL,
    PRIMARY KEY (day_event_id, day_of_week),
    KEY (day_of_week)
);

CREATE TABLE time_event_dows (
    time_event_id INTEGER NOT NULL,
    day_of_week VARCHAR(3) NOT NULL,
    PRIMARY KEY (time_event_id, day_of_week),
    KEY (day_of_week)
);

```

Is that really it? Mostly, yes. Though we need to talk a bit about indexes and about the attributes that we’ve skipped for brevity.

Most experienced database developers would look at the schema above and immediately notice that some “obvious” indexes are missing. For example, `day_events.user_id` must certainly be indexed. Unfortunately, there is no hard and fast rule on what columns (and combinations of columns) need to be indexed. That depends on how the tables are going to be queried by the application. The best book about

database indexes is called “Use The Index, Luke” (<https://use-the-index-luke.com/>). Go read it. We will also discuss indexes in a bit more detail in the book.

When we were talking about logical schema (especially in the beginning), we skipped some of the attributes, because they were very much similar to other attributes. For example, we would probably want to add the name of the user, and the column that stores the user’s password hash. Some of the data elements just don’t add anything new to this text, for example the event location, or the list of invited guests. As an exercise, you could go ahead and add the elements that we did not discuss, the ones that you’re interested in. Add a few rows to the catalog tables, fill in the contents of each cell, and then edit the schema definition above to include the missing pieces of data.

## Conclusion

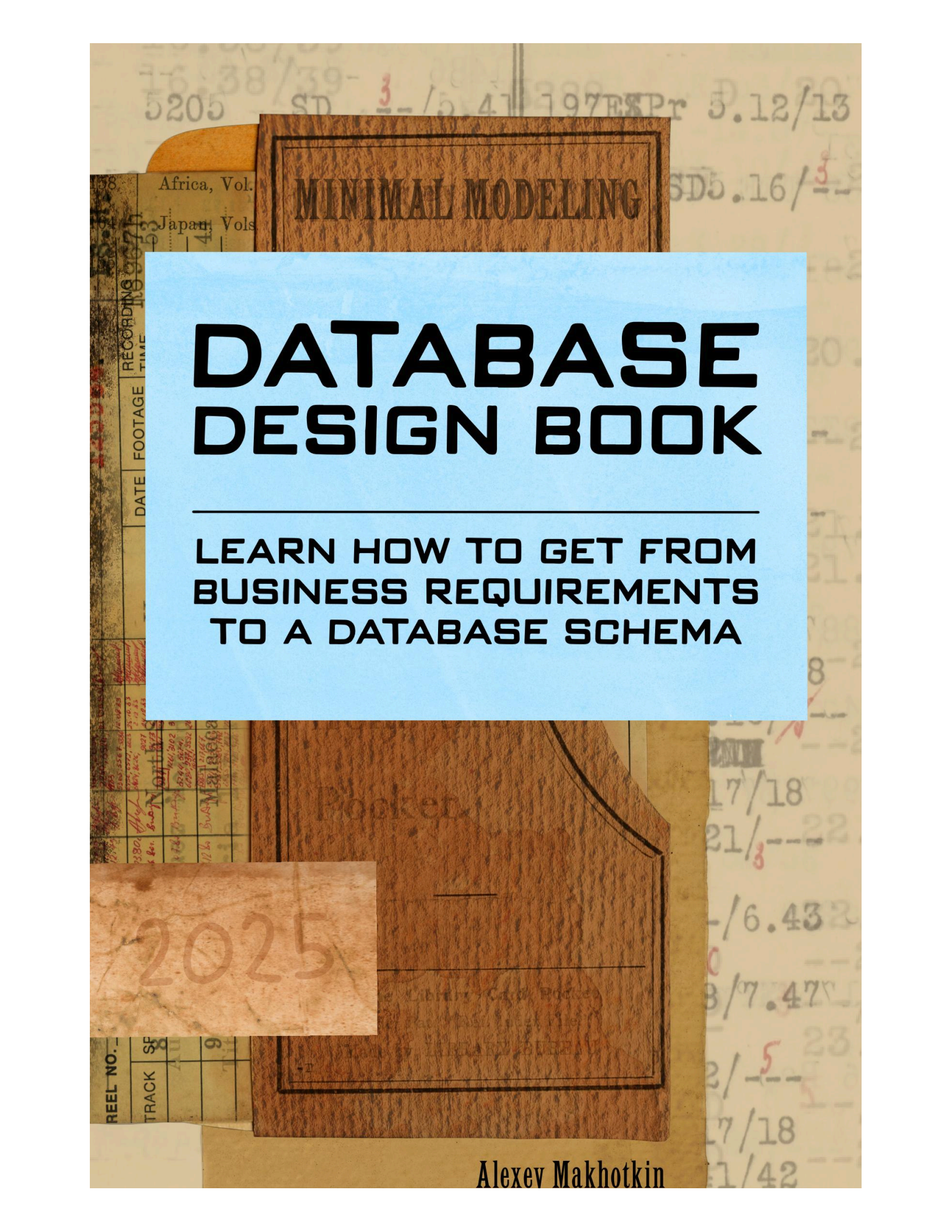
Here is a short summary of the process:

- start with a free-form text that describes the business problem you’re working on;
- write down the list of anchors, as explained above; use any collaborative tool, such as Google Docs;
- write down the list of attributes, as explained above, pay particular attention to the questions;
- write down the list of links, as explained above, pay particular attention to the sentences, because they help you make sure that you get cardinality right;
- create a graphical schema based on the logical model, if visual representation helps you think;
- fill in the physical model: table names, column names, physical data types;
- write down the SQL schema as a series of CREATE TABLE operators, using information from the previous step;
- submit the schema to your database server, fix typos, re-submit;
- share the logical model with your team;

## What’s next?

This tutorial is basically a bonus chapter from the “Database Design Book” that was released recently (<<https://databasedesignbook.com/>>).

If this tutorial helped you in understanding some aspect of database design, I’d love to hear your feedback.



# **DATABASE DESIGN BOOK**

---

**LEARN HOW TO GET FROM  
BUSINESS REQUIREMENTS  
TO A DATABASE SCHEMA**

**Alexey Makhotkin**

"449381"

4/03  
--8.16