BACK TO ARTICLES LIST

August 4, 2017 - 10 minutes read

EXAMPLE ER DIAGRAMS

Clinic Management System Data Model

Emil Drkušić
Database designer and developer, financial analyst.

Tags: database model | example data model | example ER diagram | example ERD diagram | template

Many medical clinics have shifted to an all-digital record system. What does a basic clinic management data model look like?

Visiting a hospital or a clinic is never pleasant, but it would be even worse if our health records were in chaos. Not so long ago, all medical documents were in paper

form. This not only polluted the environment, it slowed down the whole process. In some cases, patients were responsible for their own medical records. You'd see them hanging on to a sheaf of papers as they waited to be called in to the doctor.

Fortunately for us, technology has had a significant impact in the medical record field. Most health records are automated, which saves a lot of bother. Today, we'll consider a data model that could manage a medical clinic, from patient records to appointment schedules.

First, let's get an idea of what we expect from the model.

What should our data model contain?

We can figure this out in three questions:

1. What basic functionalities should this model support?

   For each patient, regardless of the reason for their visit, we should be able to:

   - Schedule or enter a new appointment
   - Create or update patients' health-related documents
   - Store the "result" of an appointment
   - Schedule follow-up appointments, if needed.

2. What additional information might be useful?

   For each visit, we could track the exact quantities of the materials and equipment used. That information would also be useful for calculating costs and invoices. We also might want to relate invoices to health insurance policies.

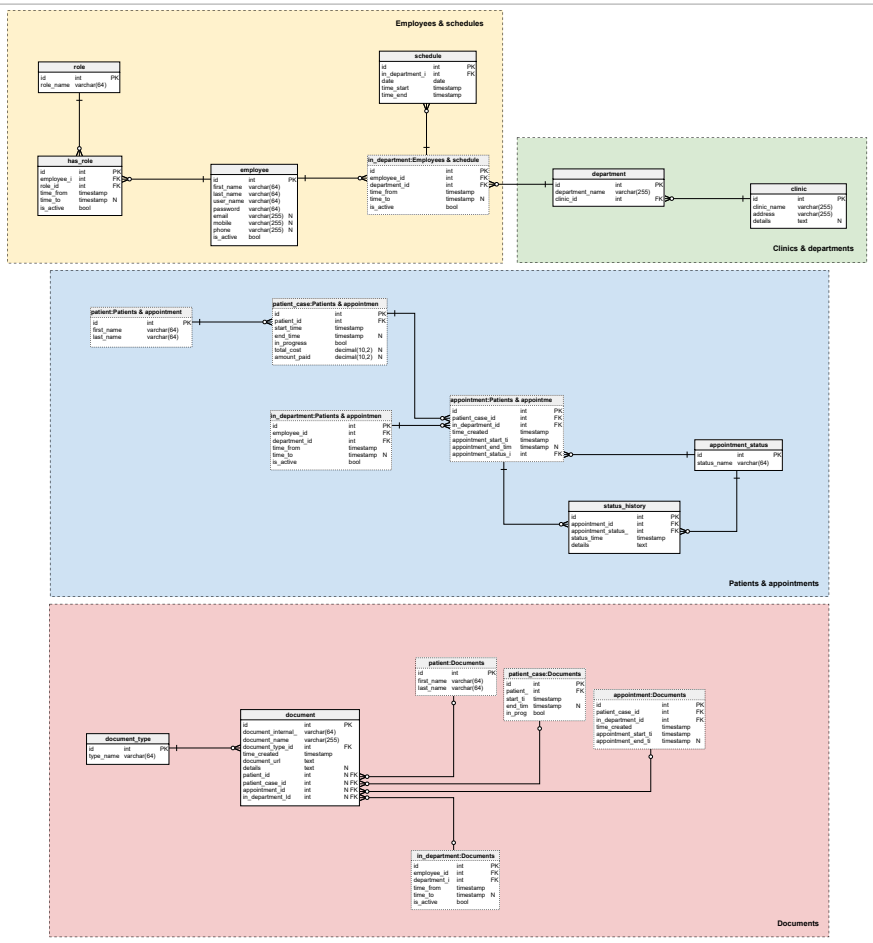
3. What aspects of the model will we focus on in this article?

   We won't go into billing and materials. They are related to the billing process and to inventory control, so they are not specific to a clinic management

system.

On the other hand, tracking appointments and their results, managing documents, and assigning staff is crucial for clinic management. We'll focus on those areas.

The Data Model



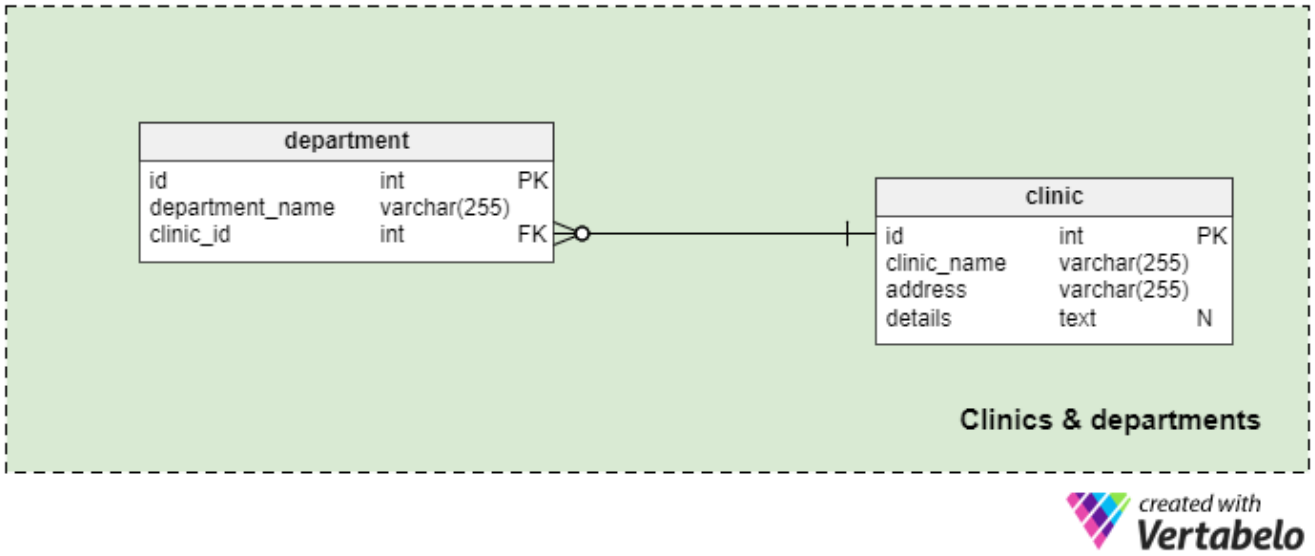
EDIT MODEL IN YOUR BROWSER

The data model consists of four main subject areas:

- `Clinics & departments`
- `Employees & schedules`
- `Patients & appointments`
- `Documents`.

We'll describe each subject area in the order they're listed.

Section 1: Clinics and departments



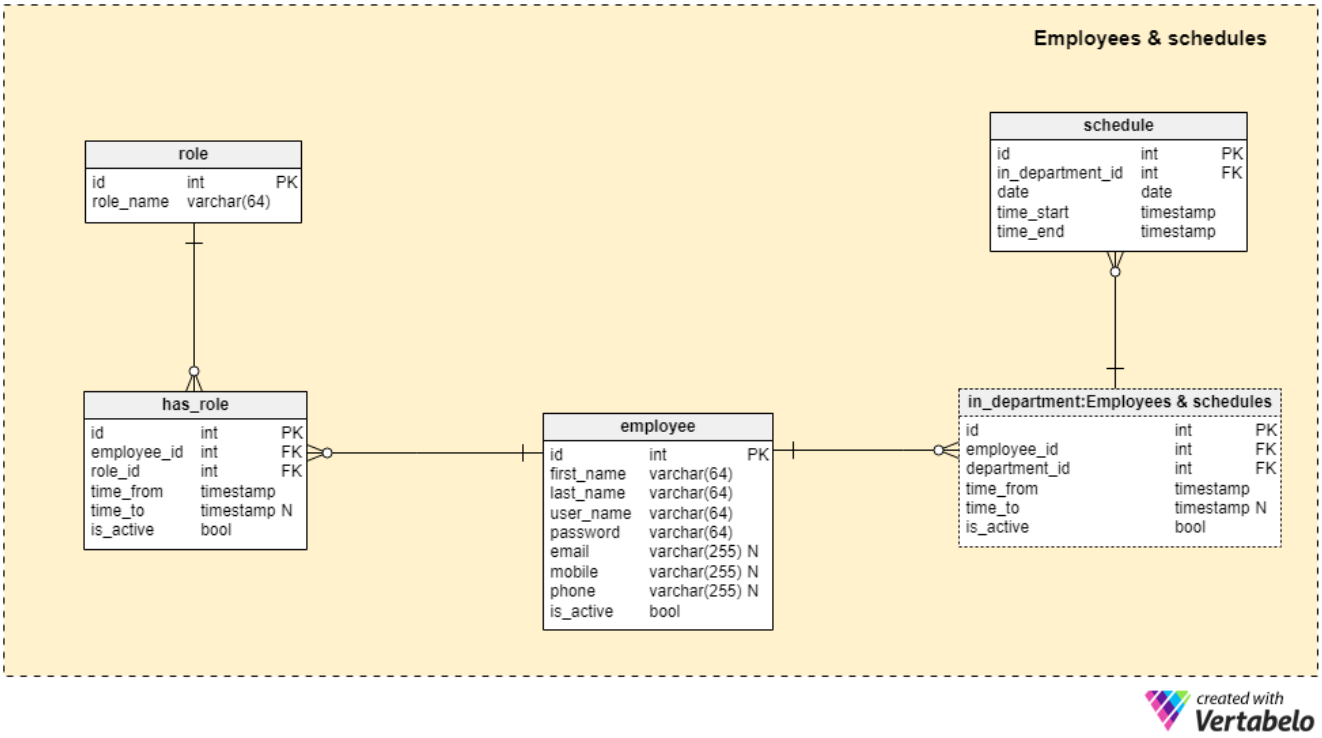
The first subject area is `Clinics and departments`. It contains two tables that store information related to the different clinics in our system and the different departments in these clinics.

The `clinic` table lists all clinics we operate. The `clinic_name` is the UNIQUE attribute while attributes `address` and `details` attributes are used to store the location of that clinic and any other information in a textual format. This table is very important when we operate more than one clinic.

The second table in this subject area is the `department` table. This is the place where we'll store the different departments of each clinic. Each department is UNIQUELY defined by its `department_name` and the ID of the clinic it belongs to.

Some well-known departments are “Emergency Medicine”, “Immunology”, “Internal Medicine” and many others.

Section 2: Employees and schedules



Facilities, departments, and equipment are necessary for running a clinic, but they mean nothing without employees – doctors, nurses and all others. To ensure a well-run clinic, we need to store employees’ personal data, roles, and schedules. This is the function of the five tables in the `Employees and schedules` subject area.

The central table here is the `employee` table. This holds a list of all employees working in any of our clinics, no matter what their role. For each employee, we’ll store:

- `first_name` & `last_name` – Are the first and the last name of that employee.
- `user_name` – A UNIQUE value the employee will use to access our system.
- `password` – A password the user will use to access the system.
- `email`, `mobile` and `phone` – Are arbitrary contact details we’ll store for users. We can expect that we’ll have at least one of them in our system.

- `is_active` – Denotes if the user is currently active in our system. This flag is set to True if the employee currently works in the clinic and false otherwise.

Two tables in this subject area are used to relate users with roles.

The first is the `role` dictionary. It contains only UNIQUE `role_name` values. We can expect that roles will be closely related to positions in the clinic (i.e. doctor, nurse, medical assistant). Note that role-related user permissions are not stored in the database; we would test permissions on the application's front end.

The second table, `has_role` , stores relationships between employees and roles. Each record in this table must contain foreign keys from both the `employee_id` and the `role_id`. We'll always store the `time_from` timestamp, as it denotes when each user started this role. We store the moment they stopped performing a role in the `time_to` timestamp. This value could be set in advance (e.g. hiring a temporary employee for a six-month period) or it could be open-ended (e.g. a regular hire). I have made this attribute NULLable so that it can accept either situation. The last attribute in this table is the `is_active` attribute, which denotes if this employee-role pairing is active or inactive. It will be set to "False" when the `time_to` value has passed. The combination of the `employee_id` - `role_id` – `time_from` attributes forms this table's UNIQUE key. We should programmatically check for overlapping when adding or changing records. We shouldn't assign the same role to the same employee for the same time interval more than once.

The remaining two tables in this subject area are intended to store details related to employees' departments and schedules.

The `in_department` table has almost exactly the same structure as the `has_role` table. It has two foreign key attributes and a time range when this data is valid. The `is_active` attribute also has the same purpose. In this table, the `employee_id` – `department_id` – `time_from` combination forms the UNIQUE key of the table. Again, we should check for overlapping time intervals when data in this table is added or changed. The idea is that each employee should work in only one department at a

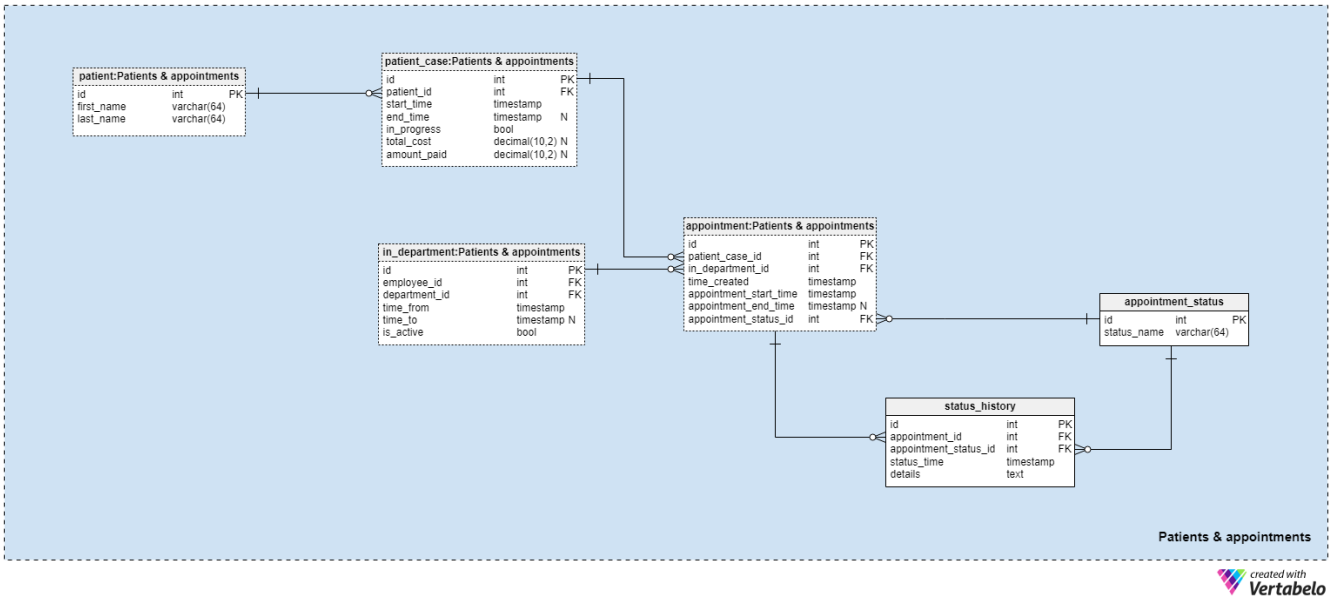
time. This model allows us to assign the same employee to more than one department, but we should only do that in special circumstances.

The `schedule` table is the last table in this subject area. It looks simple, but it's very important. Schedules should be defined upfront so that administrators can effectively plan staff for the week or month. Working in an environment where no-one knows what to expect is very stressful on everyone, and very hard for the managers to organize. This table aims to prevent this situation. (Of course, there will always be last-minute changes, but we can cope with a few of those.) In this table, we'll store:

- `in_department_id` – indirectly denotes the employee and the department.
- `date` – A scheduled date.
- `time_start` and `time_end` – A time interval when the employee starts and ends a shift. Both of these values are timestamps; we can expect that some shifts will start on one day and end another day (i.e. overnight shifts). The `time_start` will be the same as its `date` but `time_end` date could differ.

I've used the `in_department_id` – `date` and `time_start` combination as the UNIQUE key for this table. Using the `in_department_id` – `date` pair wouldn't cover employees with split shifts (e.g. a 4-hour morning shift and a 4-hour shift that evening).

Section 3: Patients and appointments



We'll use tables from the `Patients and appointments` subject area to store patient details and activities during clinic visits. Note that the `in_department` table is just a copy. It's used here to reduce overlapping relations and to simplify the model. Although you might think that the tables in this subject area are not related to any others, this table relates them.

Patients are the most important part of our clinic because there would be no need for a clinic without them. For each `patient`, we'll store only their first and last name. We could have other details as well, but these would add no value to this model. Besides, you could always add attributes to this table for those details.

Next in importance is the `appointment` table. Patients usually expect to make an appointment to go to the clinic or see the doctor. We'll store the clinic location, the patient, the department, and the employees involved in the appointment. Each record will contain:

- `patient_case_id` –Relates this appointment to a patient and a specific case. (We'll describe the `case` in a minute).
- `in_department_id` – The ID of the employee (usually a doctor) in charge of that appointment.
- `time_created` – When this record was created in our system.

- `appointment_start_time` and `appointment_end_time` – Defines the start time of an appointment and its duration. Please notice that the end time can be NULL because it could be entered after the appointment ends.
- `appointment_status_id` – References the `appointment_status` table and denotes the current status of that appointment, such as “scheduled” or “in progress”. If the patient wanted to postpone an appointment, we would assign a “postponed” status to this field and create a new appointment.

We’ve already mentioned that appointments are not directly related to patients. I’ve used the `patient_case` table to link the two relevant tables. The reason for that is simple. A patient could come to our clinic multiple times and for various reasons. Each reason for a visit is treated as one case. For each case, the patient could have multiple appointments. For example, if someone breaks a leg and needs an operation to set it, they will have several appointments before and after the operation. The broken leg and everything related to it would be a single case. If the unlucky patient came back again with a broken arm, that would be another case. This way, we know what appointments belong to what case. The attributes in the `patient_case` table are:

- `patient_id` – References the `patient` in the case.
- `start_time` and `end_time` – Denotes the moments when this case started and when it ended. Once more, we’ll probably define the `end_time` when the case is closed; therefore, this is a NULLable column.
- `in_progress` – If this case is still active (set to “True”) or not (set to “False”). When we set the `end_time` attribute.
- `total_cost` – The total amount a patient will pay for that case.
- `amount_paid` – The actual amount paid for that case.

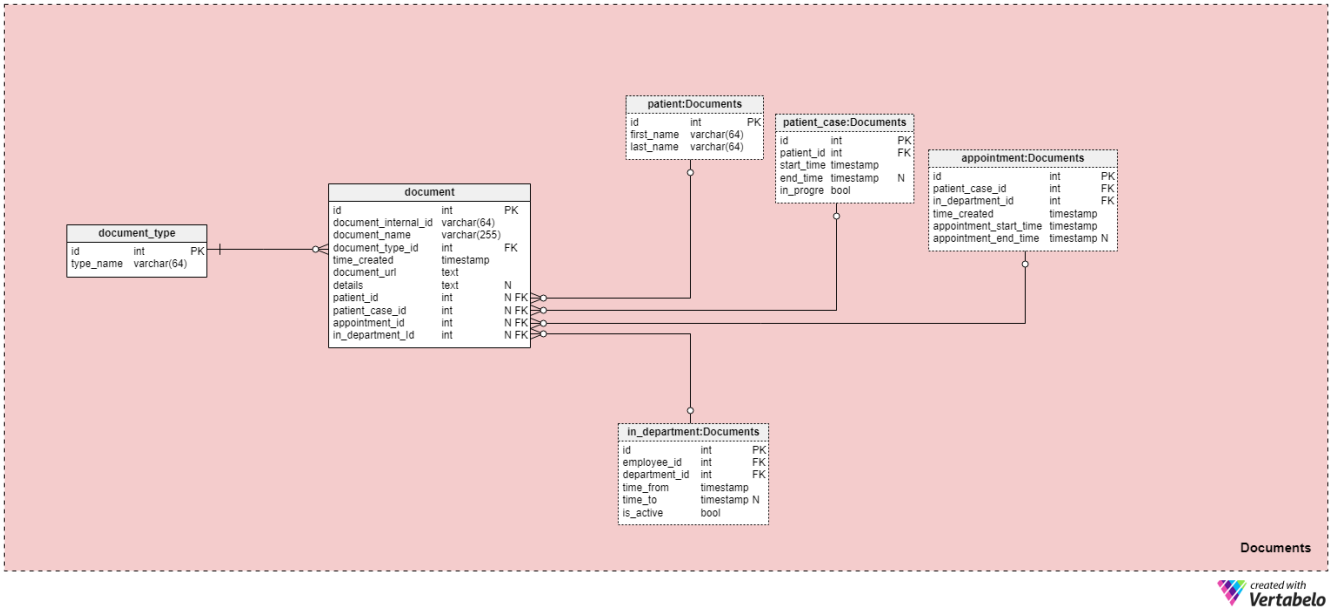
We have already mentioned the `appointment_status` table. It is a simple dictionary that stores all possible appointment statuses, such as “scheduled”, “canceled”, “postponed”, or “completed”. Status information will help us quickly filter appointments based on current status, e.g. finding all appointments scheduled for

today. Generally, each appointment will change its status from “scheduled” to another status after its start time and/or end time have passed.

We also want to store the history of appointment statuses. Each time we make a change to the `appointment`. `appointment_status_id` attribute, we’ll add one row to the `status_history` table. For each new row, we’ll store:

- `appointment_id` – The ID of the related appointment.
- `appointment_status_id` – The status ID assigned to that appointment.
- `status_time` – The timestamp when that status was assigned.
- `details` – All details, in free textual format, related to that appointment and status. This is the good place to explain what led to the status.

Section 4: Documents



The last part of our model is the `Documents` subject area. Only two of its six tables are unique to this subject area: `document` and `document_type`. The remaining tables are copies from other subject areas. They are here to simplify the model.

The `document` table contains all documents related to patients in any way. This is where we’ll store medical records, bills, and any other document type. For each document, we’ll record:

- `document_internal_id` – An internal designation we'll use to UNIQUELY denote that document.
- `document_name` – A name we have chosen for that document.
- `document_type_id` – A reference to the `document_type` dictionary.
- `time_created` – A timestamp when this document was created.
- `document_url` – An address of the location where the document is stored.
- `details` – Are all details used to closely describe this document.

The remaining four attributes are foreign keys that reference records related to that document. All of them are NULLable, but we'd expect at least one of them will be NOT NULL.

We have presented a data model that could be used to run a clinic. This is a very basic model and such systems are very complex. We could say these 14 tables are just a start. What about resource management, inventory, and payments? How would you add these features? What else would you expect to find in this data model?

Tags: database model | example data model | example ER diagram | example ERD diagram | template