**The first user group: Vet and Vet Technicians.** These users must be able to insert and read data about care being performed on the patients. This information includes vitals measurements, as well as any treatments or procedures. Vitals, such as age, weight, and blood pressure are all numerical in nature. Certain obvious constraints apply to these categories, such as they cannot be negative, age and bp are integers where weight can be a decimal number (only out to the tenths place). There also must be a few text fields with less constraints. There must be a varchar field, perhaps 30 characters, such that non-numerical treatments can be entered: Name of medicine or procedure, and administered dosage (varchar, so we don’t have to mess with unit conversions). Additionally, a large text field (1-2k varchar) must exist for important notes relating to the procedure. There are little constraints on this information, it is left as very freeform text. This field is most important for more complicated/extensive procedures. All of this treatment information must be stored, along with date information, so as to maintain a record of the patient’s health over time. The patient (animal) itself should be stored in a relation of its own and joinable to the vitals/treatments. The patient will have information such as its name, owner, species and breed. These are all varchar fields.  
  
**The second user group: Front desk, clerical, administrative** **staff.** These users are primarily involved in billing and scheduling. These users need access to the care provided to a patient, and they need to be able to join this to data that contains costs of procedures. The treatment data contains numerical information (out to the hundredths place) for the cost of a standard dosage. This treatment information is used in conjunction with the patient’s care information, allowing the cost of a patient receiving non-standard dosages to be calculated. The cost can be calculated as a fraction either in the SQL itself, or the application that the user is using. Calculating cost on the fly is a difficult task, as dosages are not all measured in the same units (pills, ounces, minutes). Either way, often the amount owed will not typically match the amount in the ‘cost’ table, as the administered dosage likely won’t match the standard dosage we have a cost stored for. This user group must have access to scheduling. Thus, there must be a data containing date and time information. Not all vet visits are the same length, so there is the concept of a ‘duration’ as well. This can be implemented as a start and end time (time data type). Constraints can be set up to prevent double-booking, although this could be tricky. The scheduling application software could also be used to protect against double booking. Finally, a information about the veterinarians themselves is necessary, as there could be multiple vets with appointments simultaneously. Vet information will be a first name (varchar), last name (varchar), and id (int, unique). In the future, this can be set up with a ‘role’ varchar attribute, in case this database was expanded in the future to include additional functionality, such as managing payroll or employee scheduling. The current implementation does not plan to service those needs, but is easily extensible in the future.

**The last user group #3: Customers**. Customers need to have access to a lot of the same information as the first two users, but in a much more restricted sense. Customers should have access to scheduling information only pertinent to their pets. In addition, they are given access to itemized receipts for the care of their pets, but do not have access to the full care notes, nor do they have access to the marginal cost of the medicines/procedures. For online billing, customers will have access to their account, which is a relation that contains customer first and last name (varchar), a unique customer id (int), and a FK join in all of the pets that they own. Additionally, the customer should have insert/update access to a relation that contains billing information. The customer should be able to store credit card information, including the CC number (varchar), expiration date (date), and the security code (varchar). Finally, there should be several attributes for billing‑street address (varchar), zip code (varchar), town and state (varchar for both). The billing information should be one to many, as one user may have several cards on file.

Combining all of my requirements: First and foremost, I did not see the need to make any compromises in terms of attribute data types, so the attributes as laid out above can stay as listed. In the first design, there will be a relation containing information about care (“chart”), which has attributes described in the first section. This relation is accessible by all three user groups, although the customers will not have access to the detailed notes section. This relation will need to be joinable to a “treatment” relation, which contains medicines and procedures, as well as a standard dosage/time, and a marginal cost. The ‘chart’ relation will need to also be joinable against the “vet” table, and the “patient” table. I will not make it joinable against the scheduling table, however, and thus the chart table will have its own date stored.

The vet table will be joinable against the schedule table as well as the appointment table. The patient table (containing information about pets) needs to contain a natural way to join it again chart, schedule, as well as customer. Customer only needs to be joinable against schedule, patient, and billing.  
 Overall, making this system work for the three user groups seems to be more of granting/restricting access, rather than converting between different domains.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Customer | | | | |
| Attribute Name | Data Type | Size | Constraints/Domain | Cascading Problems |
| cid | int | 8 | Unique, not null | ON DELETE RESTRICT  Don’t leave orphaned patients in the DB |
| fname | varchar | 50 | Not null |  |
| lname | varchar | 50 | Not null |  |
| phone | varchar | 11 | Not null |  |
| email | varchar | 50 |  |  |

**Primary Key(s):**

cid

**Foreign Key(s):**

**Indices:**

fname, lname

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| PATIENT | | | | |
| Attribute Name | Data Type | Size | Constraints/Domain | Cascading Problems |
| pid | int | 8 | Unique, not null | ON DELETE RESTRICT Patients who have had appointments should not be delete-able, as the vets office will want record of all transactions. |
| name | varchar | 50 | Not null |  |
| species | varchar | 30 | Not null |  |
| breed | varchar | 30 |  |  |
| customer\_id | int | 8 | Not null |  |

**Primary Key(s):**

pid

**Foreign Key(s):**

customer\_id REFERENCES customer(cid)

**Indices:**

customer\_id, species

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| TREATMENT | | | | |
| Attribute Name | Data Type | Size | Constraints/Domain | Cascading Problems |
| tid | int | 8 | Unique, not null | None, no tables reference this one. |
| treatment | varchar | 30 | Not null, upper, no trailing/leading whitespace |  |
| dosage | decimal |  | Not null |  |
| dosage\_unit | varchar | 15 |  |  |
| cost | numeric | 7, 2 | Positive |  |

**Primary Key(s):**

tid

**Foreign Key(s):**

**Indices:**

treatment

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| VITALS | | | | |
| Attribute Name | Data Type | Size | Constraints/Domain | Cascading Problems |
| vid | int | 8 | Unique, not null | ON DELETE RESTRICT  This should not be deleted unless the whole appointment is. |
| weight | numeric | 5, 1 | Not null |  |
| diastolic\_bp | smallint |  | Positive |  |
| systolic\_bp | smallint |  | Positive |  |

**Primary Key(s):**

vid

**Foreign Key(s):**

**Indices:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| APPOINTMENT | | | | |
| Attribute Name | Data Type | Size | Constraints/Domain | Cascading Problems |
| date | date |  | Not null | None, no tables reference this one. |
| patient\_id | int | 8 | Not null |  |
| vet\_id | int | 8 | Not null |  |
| vitals\_id | Int | 8 | Not null |  |
| treatment\_id | varchar | 30 | Not null |  |
| admin\_dosage | decimal |  | Not null |  |
| admin\_dosage\_unit | varchar | 15 |  |  |
| cost | numeric | 7, 2 | Positive |  |

**Primary Key(s):**

date, patient\_id, treatment\_id

**Foreign Key(s):**

patient\_id REFERENCES patient(pid)

treatment\_id REFERENCES treatment(tid)

vet\_id REFERENCES employee(eid)

vitals\_id REFERENCES vitals(vid)

**Indices:**

vet\_id

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| EMPLOYEE | | | | |
| Attribute Name | Data Type | Size | Constraints/Domain | Cascading Problems |
| eid | int | 8 | Not null, unique | Yes, make FK null in other tables, rather than delete tuples. |
| fname | varchar | 50 | Not null |  |
| lname | varchar | 50 | Not null |  |

**Primary Key(s):**

eid

**Foreign Key(s):**

**Indices:**

lname

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| BILLING | | | | |
| Attribute Name | Data Type | Size | Constraints/Domain | Cascading Problems |
| bid | int | 8 | Not null, unique | None, no tables references this one. |
| customer\_id | int | 8 | Not null |  |
| card\_no | varchar | 16 | Not null, numerical |  |
| exp\_date | date |  | Not null |  |
| security\_code | varchar | 4 | Not null, numerical |  |
| street\_address | varchar | 30 | Not null |  |
| town | varchar | 30 | Not null |  |
| state | varchar | 2 | Not null, upper case |  |
| zip | char | 5 | Not null |  |

**Primary Key(s):**

bid

**Foreign Key(s):**

customer\_id REFERENCES customer(cid)

**Indices:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SCHEDULE | | | | |
| Attribute Name | Data Type | Size | Constraints/Domain | Cascading Problems |
| customer\_id | int | 8 | Not null | None, no tables reference this one. Appointment cancellations are expected |
| patient\_id | int | 8 | Not null |  |
| date | date |  | Not null |  |
| time\_start | time |  | Not null |  |
| time\_end | time |  | Not null |  |
| vet\_id | int | 8 | Not null |  |

**Primary Key(s):**

patient\_id, date

**Foreign Key(s):**

customer\_id REFERENCES customer(cid)

patient\_id REFERENCES patient(pid)

vet\_id REFERENCES vet(eid)

**Indices:**

vet\_id, time\_start, time\_end

**Normalization:**

First, a new UNF relation is created, containing all of our attributes. Foreign key attributes are omitted, as they are just duplicates of the primary key attributes in the table they reference. There is one derived attribute, */cost*.

A screenshot of a cell phone

Description automatically generatedTo normalize the UNF relation, the existing attributes are atomized, and a primary key is selected. In order to atomize the data, tuples are created in order to flatten out any multivalued attributes, and any composite attributes (like name) are broken into their parts. As there are no tuples in this table, there are no multi-valued columns that need to be expanded. Additionally, this data is largely already atomic; specifically, names are already in fname/lname form. The only other attribute that could be decomposed into parts is the street address (number and street), but that level of granularity is not useful for billing information. Another consideration in the UNF is the existence of multiple attributes with the same name that contain slightly different information. In order to differentiate the attributes with duplicated names, they are renamed before moving forward. Renamed attributes are shown in blue in the 1NF relation. Finally, to achieve 1NF, a primary key is selected that uniquely identifies every tuple in the table (shown highlighted in gold). The “id” attributes compartmentalize most of the data very well, making selecting a PK fairly straightforward. The following table shows how each column is uniquely identified (noting there are multiple combinations of PK attributes any given tuple can be uniquely identified):

|  |  |
| --- | --- |
| **PK Column(s)** | **Uniquely Identifies** |
| pid | cid, c\_fname, c\_lname, phone, email, name, species, breed |
| tid | treatment, dosage, dosage\_unit, cost\_marginal |
| pid+date+eid | time\_start, time\_end, fname, lname |
| pid+date\_appt | vid, weight, diastolic\_bp, systolic\_bp |
| pid+date\_appt+tid | admin\_dosage, admin\_dosage\_unit, cost |
| bid | card\_no, exp\_date, security\_code, street\_address, town, state, zip |

Serendipitously, this table will help us migrate to 2NF. To meet 2NF, we need every one of our attributes to depend on every field of the primary key. Looking at the table above, it is clear this data needs to be broken up into several relations to meet 2NF.

While decomposing this relation into several new relations to meet 2NF, I note a previous unspoken assumption. For this design, I am assuming that any given patient will only have a single appointment in a given day. If we wanted to change that, it we would want to add *time\_start* to the *vitals* relation. That is, because we have one appointment per day per patient, *time\_start* is derivable from pid+date\_appointment, and would be removed in the next normalization step. For now, I will continue with the assumption that a patient will only have one appointment per day, maximum.   
 The second observation made more apparent through this normalization, is it might be worth refactoring out the second date attribute.

A screenshot of a social media post

Description automatically generated

Now, we can convert to 3NF. In order to meet 3NF, we have to make sure that no attribute depends on a non-PK attribute. In the 2NF, there are two relations that need refactoring. First, we note that all of the customer fields can be derived from *cid*, independent of the primary key. Thus, we can break this out into a new *customer* relation. The next dependency we note, is that *cost* is a derived attribute, dependent on the administered dosages, and the dosages in the *treatment* relation. While it meets 2NF, cost does not depend on the PK of *administered*. We will break that attribute out into a separate table as well.

A screenshot of a social media post

Description automatically generated