

Table of Contents

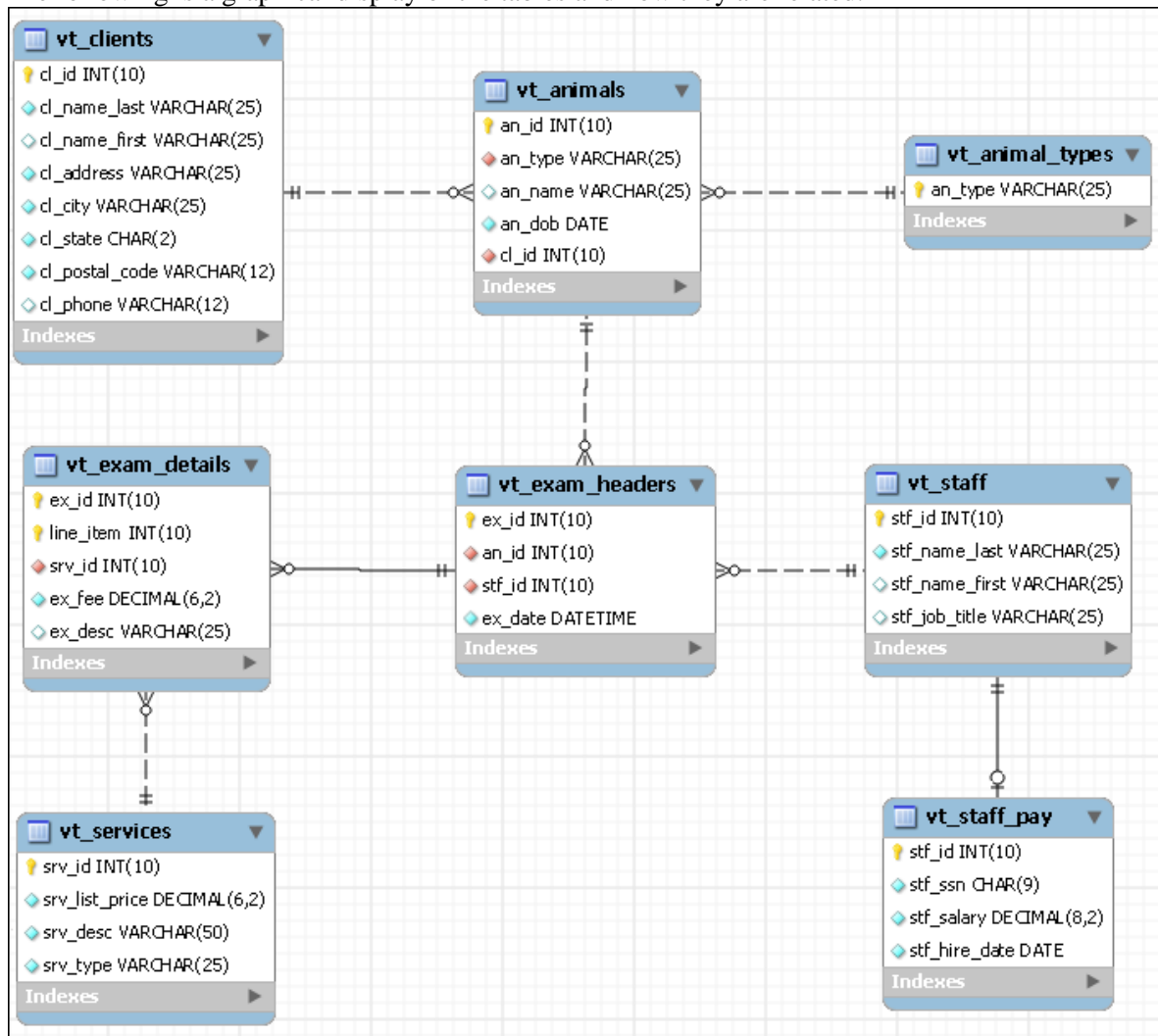
1. Tables in the vets database.....	1
2. Business Rules:	2
3. SQL Discussion:.....	2
3.1. vt_animals.....	2
3.2. vt_exam_headers and vt_exam_details	3
3.3. vt_services.....	5
3.4. Inserting data	5

1. Tables in the vets database

These are the tables for the a_vets database. This diagram was set up as a Database Diagram in MySQL Workbench.

For some assignments, you will need to use tables from the a_vets database and you need to understand how the tables work..

The following is a graphical display of the tables and how they are related.



2. Business Rules:

The business rules determine how the tables are set up. We cannot enforce every business rule via the database tables.

- Each animal seen at the Vet clinic is the responsibility of a single client.
- We need name and address information for the clients.
- The vets treat only certain type of animals.
- For the staff at the clinic, we are only storing part of their data in this database- specifically we are not storing their payroll details info here. There are two staff tables- one is `vt_staff` with their public information and a second table, `vt_staff_pay`, with information that would be more private.
- We have a list of the various services that the clinic can provide and we have a list price for each service.
- When an animal comes in for an exam we keep track of some of the exam data in this database- primarily the data that might be used for billing.
- This database does not handle payment of the bills.

3. SQL Discussion:

This discussion will take you through some of the details in a Create Table statement. You do not need to remember all of these details yet but you should be able to understand these. It will help you understand the tables if you read through the Create statements.

Some MySQL specific features:

- Depending on your OS, the table names may be case specific. I am using all lower case letters for the table names. This is not an issue with the Windows O/S but can be with a Linux Os
- I end each of the create statements with the phrase `engine = INNODB`. This ensures that we have transactional tables; don't worry about that now. This is the default in my installation of MySQL but not in all installations.
- I am including check constraints to show you what types of constraint rules you might want to use. Unfortunately, at this time, MySQL does not enforce check constraints.

3.1. `vt_animals`

Let's start with the animal table. For each animal, we want to know what type of animal it is- a cat, dog, turtle etc. We also want to know its name- but we do not insist that it has a name. We want a date of birth for each animal to calculate its age. If necessary, the vet will estimate this date. Since we are storing data for a veterinary clinic we need to know who will pay the vet bills- so for each animal we will also store the id of the client responsible for this animal. And we need an identifier for the animals.

This is the create table statement for the animals table. I added line numbers to make this easier to discuss but the line numbers are not part of the SQL.

```
1. create table vt_animals(  
2.     an_id          int unsigned not null  
3.     , an_type      varchar(25) not null  
4.     , an_name      varchar(25) null  
5.     , an_dob       date      not null  
6.     , cl_id        int unsigned not null  
7.     , constraint vt_animals_pk primary key (an_id)  
8.     , constraint vt_animals_animal_types_fk  
9.         foreign key (an_type) references vt_animal_types (an_type)  
10.    , constraint vt_animals_clients_fk  
11.        foreign key (cl_id) references vt_clients (cl_id)  
12.    , constraint an_id_range check (an_id > 0)  
13. ) engine = INNODB;
```

The statement starts by saying we want to create a table and supplies a name for the table (line 1)

The statement lists the various attribute names and their data type (lines 2-6). "Int" is an integer number; int unsigned means the numbers cannot be negative; "varchar" is used for character data and includes a maximum length- in this case the animal name must be 25 characters long or shorter; "date" is used for date values- these store the date without a time component.

We also indicate if a value is required to be entered for each attribute. "**not null**" means that we have to supply a value and "**null**" means that the value is optional. Note that the an_name column is designated as "null" and the other columns as "not null".

Then we have some constraints defined for the various attributes.

The phrase "**primary key**" (line 7) is used to indicate that this column is the identifier for the table. Designating a column as a primary key means that it cannot be null and it must be unique in this table. Each animal in the table has to have a different id value.

There are two "**foreign key ... references**" phrases (lines 8 and 9). The vet is not willing to treat all types of animals- for example our vet will not treat fish. So we have a table vt_animal_types that includes the allowed values for an_type. The references phrase in the animals tables enforces the rule that any value entered for the an_type column must be a value in the vt_animal_types table. If the vet later decides to treat fish, we can add that value to the vt_animal_types table. This has a second purpose in that we can restrict the spelling of the values entered- so a dog gets the value "dog"- not "dogs", "puppy", "canine", "poodle" or other terms. This helps keep our data more organized.

The other references clause is used to be certain that each animal does have an associated client.

Line 10 adds a traditional sql check constraint that the an_id values have to be positive values. MySql does not enforce this constraint.

Now let's think about bringing an animal into the vet clinic for an exam. We will need to have data for the animal and its client in our tables- this is sometimes called registering the animal and client. The vet will keep some information about the exam- such as the date and time of the exam, which staff person did the exam and what was done for the animal. These tables are not keeping all of the exam details but it is quite likely that an exam will include more than one treatment- and more than one billing item.

We want to keep track of the various treatments the animal has on an exam and a standard way of dealing with this is to have two tables: one table for the exam includes those attributes that occur only once per exam (the animal identifier and the date) and another table that can have one row for each treatment provided. Then we need to associate those two tables.

3.2. vt_exam_headers and vt_exam_details

This is the Create Table statement for the exam header table which stores the information for the exam. We need an identifier for the exam (line 6) and each exam is for one animal- that uses the an_id (line 3) which must refer to an animal that we have in the animal table (line 7). This means that we cannot store exam information about an animal that we do not have in the animals table.

We also have a value for the staff person who did the exam which refers to a person in the staff table (line 8). The exam date is stored as a DateTime field which will store both the date and the time of the exam. The exam date has to be no earlier than Jan 1, 1995- the date that the clinic opened (line 10- which is not enforced in MySQL) All of these fields need to have an entry.

```
1.  create table vt_exam_headers(
2.      ex_id          int unsigned not null
3.      , an_id        int unsigned not null
4.      , stf_id       int unsigned not null
5.      , ex_date      datetime      not null
```

```

6. , constraint vt_exam_headers_pk primary key (ex_id)
7. , constraint vt_exam_headers_animals_fk
   foreign key (an_id) references vt_animals (an_id)
8. , constraint vt_exam_headers_staff_fk
   foreign key (stf_id) references vt_staff (stf_id)
9. , constraint exam_id_range check (ex_id >0)
10. , constraint exam_date_range check (ex_date >= date '1995-01-01')
11. )engine = INNODB;

```

This is the Create Table statement for the exam detail table which stores the detailed information for the exam.

Try reading the SQL first and determining the rules the sql enforces..

```

1. create table vt_exam_details(
2.   ex_id          int unsigned not null
3.   , line_item    int unsigned not null
4.   , srv_id       int unsigned not null
5.   , ex_fee       numeric(6,2) unsigned not null
6.   , ex_desc      varchar(25) null
7.   , constraint evt_exam_details_pk primary key (ex_id, line_item )
8.   , constraint evt_exam_details_headers_fk
   foreign key (ex_id) references vt_exam_headers (ex_id)
9.   , constraint evt_exam_details_services_fk
   foreign key (srv_id) references vt_services (srv_id)
10. , constraint line_item_range check (line_item >0)
11. , constraint ex_fee_range check (ex_fee >= 0)
12. )engine = INNODB;

```

We have one row in this table for each treatment that was done during an exam. So we need to be able to associate a row in the table with the exam it belongs to; this is done by including the exam identifier in this table and referencing that attribute to the exam header table.

But now we cannot make that attribute the primary key for this table since we will generally have multiple rows for each exam. In this table design we are including a second attribute - the line_item which is part of the primary key. We expect that the first item for an exam will be line_item 1 and the second line_item 2- but the design does not enforce that rule. We are declaring the combination of these two attributes (ex_id and line_item) to be the primary key (line 7)

We include the treatment identifier- the service id- in the detail table and associate it with the services table. The exam detail table includes an attribute for actual fee charged. This could be the same as the list price in the services table- but the vet could choose to charge less than the list fee- or more.

This shows two rows that could be found in the exam headers tables and the related rows from the exam details table.

```

+-----+-----+-----+-----+
| ex_id | an_id | stf_id | ex_date |
+-----+-----+-----+-----+
| 1982  | 21001 | 15     | 2011-10-23 10:30:00 |
| 1988  | 15001 | 25     | 2011-11-09 09:00:00 |
+-----+-----+-----+-----+
2 rows in set (0.06 sec)

```

```

+-----+-----+-----+-----+-----+
| ex_id | line_item | srv_id | ex_fee | ex_desc |
+-----+-----+-----+-----+-----+
| 1982  | 1         | 523    | 60.00  | yearly checkup |
| 1982  | 2         | 110    | 50.00  | NULL          |
| 1988  | 1         | 104    | 30.00  | check up      |
| 1988  | 2         | 1002   | 25.00  | NULL          |
| 1988  | 3         | 687    | 45.00  | shell softening |
+-----+-----+-----+-----+-----+
5 rows in set (0.01 sec)

```

3.3. vt_services

The Create Table statement for the services has an additional rule: There is a column `srv_type` and we want to restrict the values to 4 possible values. In this table this is done with a check constraint which lists the allowed values (line 7- this is not enforced by MySQL). This is a different technique than was used for the animal types and is useful when we have a short list of possible values that is not apt to change frequently. You can see this technique also used in the staff table.

```
1. create table vt_services(  
2.   srv_id          int unsigned not null  
3.   , srv_list_price numeric(6,2) unsigned not null  
4.   , srv_desc       varchar(50) not null  
5.   , srv_type       varchar(25) not null  
6.   , constraint vt_services_pk primary key (srv_id)  
7.   , constraint srv_type_ck check (srv_type in  
      ('office visit', 'consult', 'medicine', 'treatment'))  
8. ) engine = INNODB;
```

MySQL has a different way of handling the rule that we want to limit the values for `srv_type`; we will see this later in the semester.

3.4. Inserting data

Because our tables are related to each other the inserts have to be done in a specific order. Suppose you are a new client to the vet clinic and you are bringing in your dog "mittens" for a checkup. The person registering you will need to store your client information first in the `vt_clients` table before they can store the animal's information. This is because in the `vt_animals` table, there is a required `cl_id` value that has to match a `cl_id` value in the client's table. And the exam data cannot be stored until there is a related animal row.

The vet clinic would be using an application that handles these details and the person at the registration desk would not need to know those details. But when we enter the data using our inserts statements we have to insert these in the proper order. The inserts script uses the following order for the inserts

1. staff
2. staff_pay
3. services
4. animal types
5. clients
6. animals
7. exam headers
8. exam details

Take a look at the delete statements at the top of the script and you can see that these delete statements are written in the opposite order. This is because the way the database is set up we cannot delete a parent row if there are related child rows. For example we cannot delete a client if the client has animals in the animals table. In our database we do not want to store data about animals for which we do not have a client to pay the bills!