Table of Contents

1. Two table join 1
2. Inner join 2
3. The vets database relationships 2
4. Deciding on tables to use in a query. 3

In unit 2, we looked at the vets database and its collection of tables. Hopefully by now you have noticed that these tables work together to store the data. The first time you work with a relational database it might not make sense that we have so many tables. We have a table for animal data, another for client data and two tables that together hold the exam data; we also have tables for the staff and services. We will talk more about the theory behind having all of these tables later in the semester, but for now I want to focus on how we associate the data in these tables.

1. Two table join

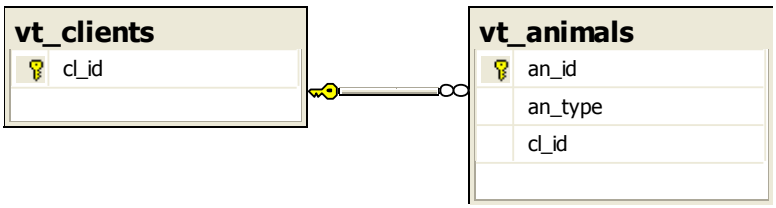
Let's start with two tables- the clients and the animals. One of the rules the vet has for his clinic is that every animal that he treats has to have an associated client- the vet might suggest treatments for the cat Mittens, but the client makes decisions about whether or not Mittens gets her teeth cleaned and it is the client who ends up paying the vet bills for Mittens. So the database tables need to have a way to go from the information for Mittens to the client responsible for Mittens.

In database terms, we need to establish a relationship between the clients table and the animals table. Commonly we do this when we set up the tables by defining primary and foreign keys.

This is **part** of the definition of these two tables- I am including only the key columns in each table.

```
create table vt_clients(
    cl_id           int         not null  primary key
);

create table vt_animals(
    an_id           int         not null  primary key
  , an_type         varchar(25) not null
  , cl_id           int         not null
, constraint fk foreign key(cl_id) references vt_clients(cl_id)
);
```



(The diagrams were done with MS SSMS since it makes nicer diagrams for this discussion.)

In this relationship, the clients table is called the parent table and each client row is identified by the cl_id. The cl_id attribute is the primary key for the clients table.

The animals table is called the child table; each animal row is identified by the an_id. The an_id attribute is the primary key for the animals table. And each animal row also has a value of a cl_id.

The clause `references vt_clients` states that the value in the cl_id column for an animal row has to match a value for cl_id in the clients table. Also this column is said to be Not Null so each animal has to have an associated client. The cl_id attribute is the foreign key to the clients table.

Because we have set this rule about animals and clients when we defined the tables, the dbms will not let us enter a row for an animal unless it has a value for cl_id that matches an existing client. The dbms also will not let us delete a client row if that client has associated animal rows. (If we could delete a client and leave the animals rows- they would be called orphaned animal rows- and we don't want that. Who would pay the vet bills?)

You would, quite logically, think that since we set up that relationship and the dbms knows about it that you could just then write SQL that knows the relationships without being told. But that is not the case. When you write a query that want to see information about an animal and information about the client for that animal, your SQL statement has to define the relationship between the tables- which is called a join in SQL.

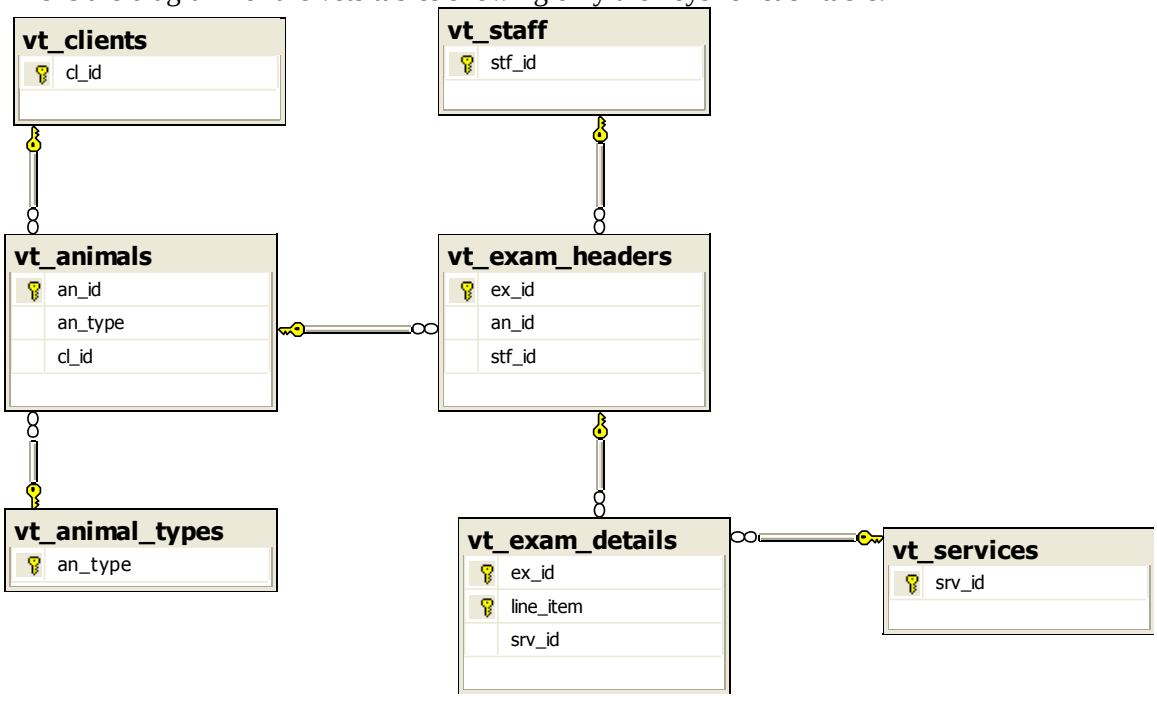
Historically, years ago, a database was commonly set up without defining the relationships in the table definitions and so SQL developed needing you to define those joins. We can also write joins that use columns other than the PK and FK fields.

2. Inner join

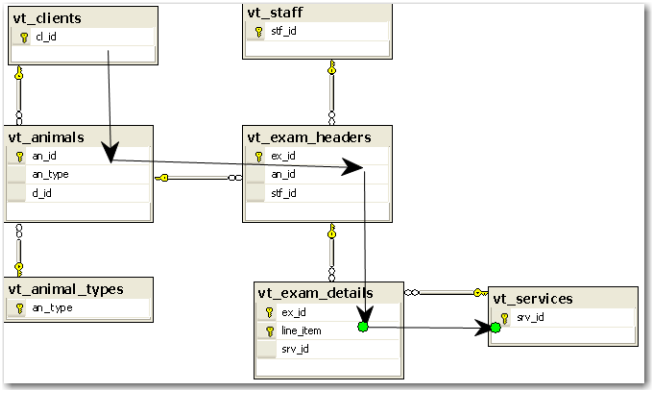
We have several types of joins possible between tables. The first one we work with is called an inner join. An inner join joins two tables on a column in each table that have values in common. A row is included in the result set if each of the two tables have matching rows. If we do an inner join between clients and animals then we get results back only for clients who have animals. In a few weeks we will also look at outer joins which would let us get results back for clients who have animals and for clients who don't have animals.

3. The vets database relationships

This is the diagram for the vets tables showing only the keys for each table.



If you think of the relationship lines on the diagram as pathways between the tables (as possible joins) then you can navigate from the clients table down to the services table and see, for each client, which services his pets has received.



Showing just the table name, the query would look like this

```
select . . .
from vt_clients
join vt_animals
join vt_exam_headers
join vt_exam_details
join vt_services
```

4. Deciding on tables to use in a query.

When you are writing a query it helps to look at the diagram and find a path between all the tables you need and then write the From clause starting at one end of that pathway and proceed one table at a time.

Sometimes people have trouble deciding which tables to include in a query. This is a technique you might try.
A: Suppose I said to write a query showing the last name of each staff person and the name of the services they performed on an animal and the date they did the service. First make a list of the attributes you need to display or filter for. Here we do not have a filter and we need to display

last name of staff

service description

exam date

Now list the tables these attributes appear in. This one is fairly easy since each of these attributes appear in only one table

Attribute	stf_name_last	srv_desc	ex_date
Table	vt_staff	vt_services	vt_exam_headers

Do we have a direct path between these tables?

```
From vt_staff
join vt_exam_headers
join vt_services
```

We can go from vt_staff to vt_exam_headers. We do not have a direct path from vt_exam_headers to vt_services, so we need to also include the vt_exam_details table

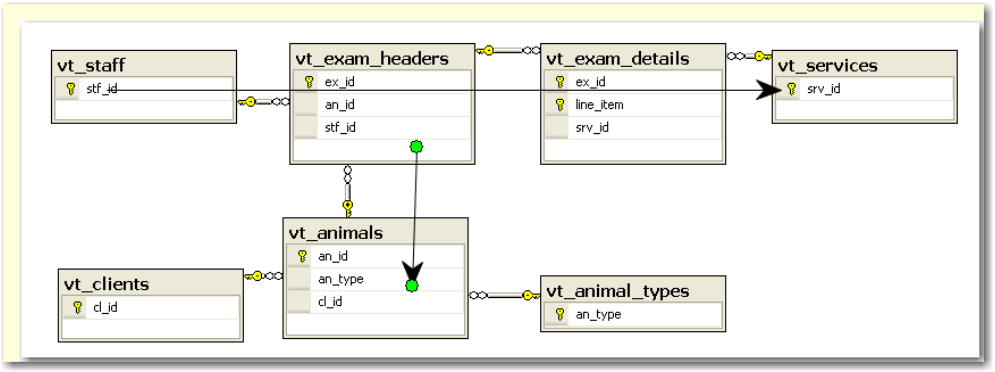
```
From vt_staff
join vt_exam_headers
join vt_exam_details
join vt_services
```

B: now suppose I changed the query and said that I only wanted to see the services if they were performed on a Bird. We need to include the an_type attribute in the Where clause. an_type appears in two tables: vt_animals and vt_animal _types

Attribute	stf_name_last	srv_desc	ex_date	an_type
Table	vt_staff	vt_services	vt_exam_headers	vt_animals
				vt_animal_types

In this case we can use the attribute in the vt_animals table since that will let us know if an animal rows is a bird or not. There is no need to include the vt_animal_types table.

Now we have a divided path



But we still have a path- we can travel from the exam headers to the animals table.

```
From vt_staff
join vt_exam_headers
join vt_exam_details
join vt_services
join vt_animal
```

C: Suppose I asked you to show me the animal id for any animal ever seen by the staff person with the staff id 103. We need to see the an_id which is in two tables and we need to filter on the staff_id which is also in two tables.

Attribute	stf_id	an_id
Table	vt_staff	vt_animals
	vt_exam_headers	vt_exam_headers

At first you might think of using

```
From vt_staff
join vt_exam_headers
join vt_animal
```

Do we need all three tables in this query? No- the data that we need is all in the table vt_exam_headers and we do not need any join.

Every extra table you add to a join makes the query slower so you want the From clause to be a short as needed (but no shorter!)