Dave Babler

Supervised by: Professor Larry Bross

**Veterinary DATABASE: REQUIREMENTS implementation & proof of concept**

# TABLE OF CONTENTS

[DOCUMENT PURPOSE AND STRUCTURE 3](#_Toc519361971)

[WARNING 3](#_Toc519361972)

[INSTALLATION 3](#_Toc519361973)

[DATA STRUCTURE CREATION 6](#_Toc519361974)

[Tablespaces 6](#_Toc519361975)

[Tables and Constraints 7](#_Toc519361976)

[Sequences and Identity Columns 10](#_Toc519361977)

[Data Loading 11](#_Toc519361978)

[BUSINESS RULES IMPLEMENTATION 12](#_Toc519361979)

[Record Keeping, Reception, Customer Relations Managment 12](#_Toc519361980)

[Clinical Procedures 19](#_Toc519361981)

[Pharmacology, Pathology, Blood Bank 24](#_Toc519361982)

[RX-10 a separate id for each individual prescription external to the chart 27](#_Toc519361983)

[RX-02 Stock reorder flag 27](#_Toc519361984)

[Patient Chart 29](#_Toc519361985)

[REPORTS REQUIRED 35](#_Toc519361986)

[MISCELLANEOUS TRANSACTIONS, REPORTS, AND VIEWS 41](#_Toc519361987)

[TRX-03 Inventory Disposable equipment 41](#_Toc519361988)

[Blood Bank 41](#_Toc519361989)

[Not included rules 42](#_Toc519361990)

[PERFORMANCE TUNING 42](#_Toc519361991)

[DATA STRUCTURES AND PROGRAM UNITS 43](#_Toc519361992)

[Tables 43](#_Toc519361993)

[BIBLIOGRAPHY 44](#_Toc519361994)

[ACKNOWLEDGEMENTS 45](#_Toc519361995)

[APPENDIX A: SQL FILES 46](#_Toc519361996)

# DOCUMENT PURPOSE

The purpose of this document is to demonstrate the implementation of the requirements set forth in the previous document Case Study for Developing A Veterinary Database (Babler, 2018). This will be done by showing the implementation of various data structures (tables, views, etc.) and program units (procedures, functions, etc.) that adhere to the business rules as set in Babler’s Case Study for Developing a Veterinary Database [CSFDAVD].

# WARNING

The scope of this paper is: Oracle PL/SQL, SQL, relational algebra, relational theory in general, data structures, structured programming, Oracle’s 12c database, anything tangentially related to these topics.

The author of this paper *is not* a veterinary scientist. There is data in the example database showing real-life veterinary pharmaceuticals and veterinary clinical procedures. These examples show how veterinary data can be used with relational objects for the efficient storage, retrieval, updating, and manipulation of data for a small to medium veterinary practice.

***Under no circumstances should the prescriptions, medicines, or clinical procedures shown in the database be administered to any animal without the explicit consent and observation of a qualified veterinary doctor.***

# NOTE

All images have been linked to files online. To see a larger version of the image, simply use <Ctrl> + Click in Microsoft Word to open it in your browser. You may also right click the image and choose “Open Link” to view the image in a browser.

# INSTALLATION

The installation of the prototype database [BablerVet] was done performed using Oracle’s Database Creation Assistant [figures 1 -3]. The memory and storage requirements were greatly reduced for this prototype to allow for simulating stresses on the database that would happen with multiple users and to save on prototype development costs.

The primary tools for administering the database were Oracle’s SQL Developer and Oracle’s SQL+; special thanks to text editors Visual Studio: Code by Microsoft and Sublime Text 3 by Sublime HQ Party Ltd.

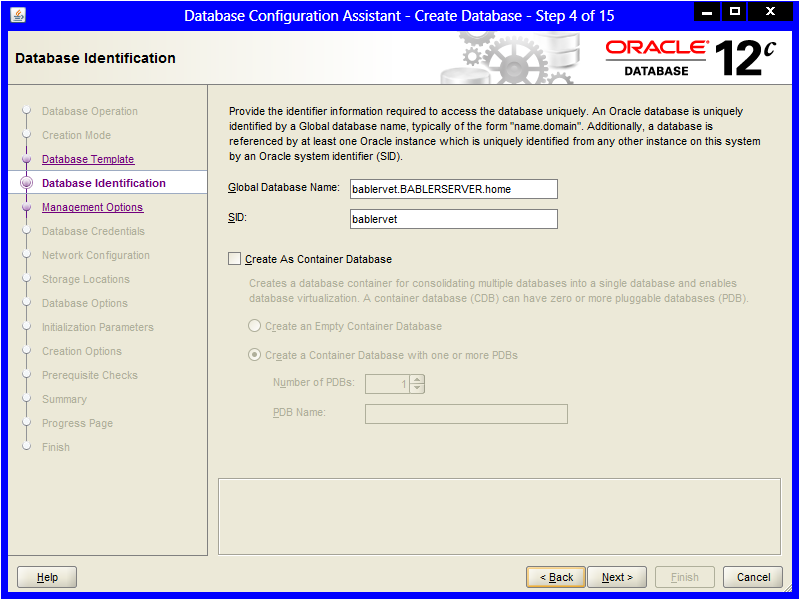
[](https://drive.google.com/open?id=1wMlQ5XdY5jkWAkY-da4BKDEOPqJYjtl1)

Figure 1

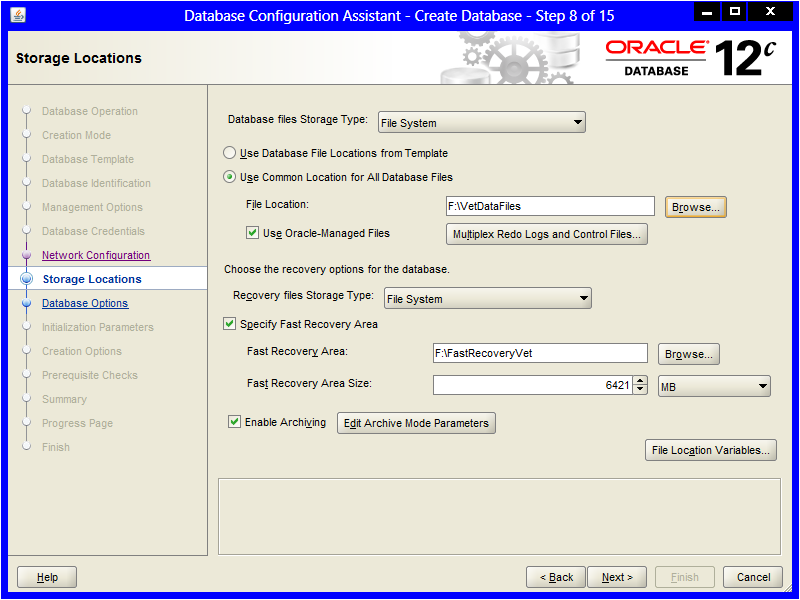
[](https://drive.google.com/open?id=1df7WocTjV8p7xbNkTMlj9Zfl_feYpXQ0)

Figure 2

[](https://drive.google.com/open?id=1q3fgqH-ifkaN1kPGgdYymOdrjk8jGrIF)

Figure 3

In a live environment various pseudo-user would be created and those pseudo-users would have their schemas shared with appropriate real users; however, for the purposes of the prototype [hereon: BablerVet] a single schema/user was wrapped up into a database administrator [DBA] account [figures 4 & 5].

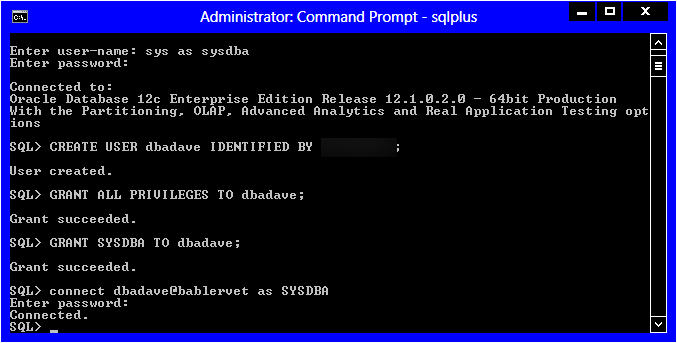
[](https://drive.google.com/open?id=1vv9n3M84h9qOu7zVNgWyb_XOPU5QeFzj)

Figure 4

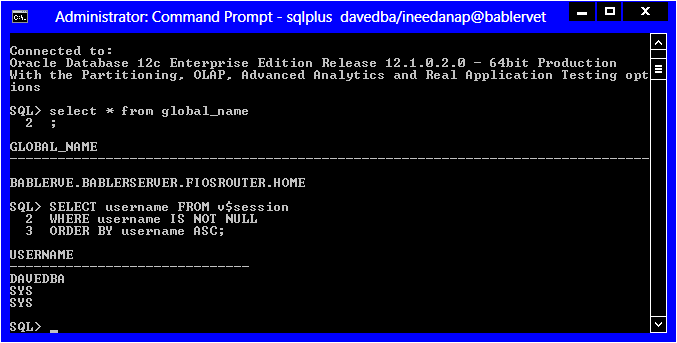
[](https://drive.google.com/open?id=11e4gRLXgM88yuYxgXAKBLh6bFLApEEc2)

Figure 5

# DATA STRUCTURE CREATION

## Tablespaces

Each general area of the business was given its own tablespace and datafiles. This was done due to discoveries in requirements analysis regarding potential regulatory issues surrounding pharmacological and personal data of patients/owners. Examples are shown in figures 6 and 7.

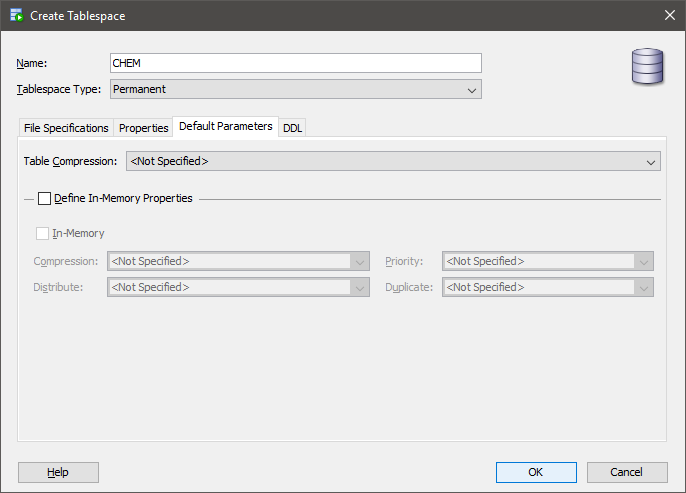
[](https://drive.google.com/open?id=1CJ_4Tyen9oldoZW3y360CyHyEWYMcTsD)

Figure 6

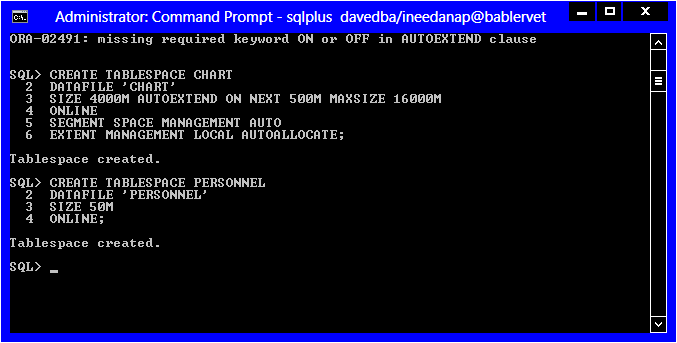
[](https://drive.google.com/open?id=1vJbqhdLpK9X-l8HNjpnPqWWZV5bAwOZ2)

Figure 7

The temporary tablespace was also dramatically increased from its default parameters. This is to accommodate all the sorting, queries, temporary tables, and complex calculations that may need to be done throughout the normal use of the database [figure 8]. The temporary tablespace was also set to expand as needed with a 12GB cap.

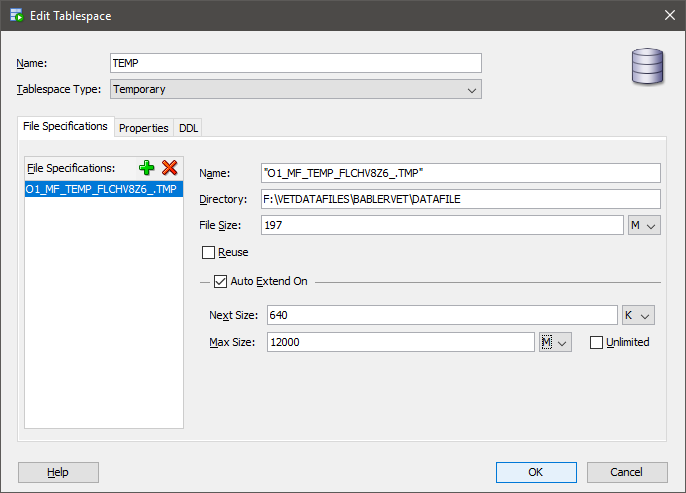
[](https://drive.google.com/open?id=17yn2MR6hrkt58wOeu7PClWbvKljZM02w)

Figure 8

## Tables and Constraints

Most of the table structures were written out in a text editor and then uploaded into the database using SQL [figure 9].

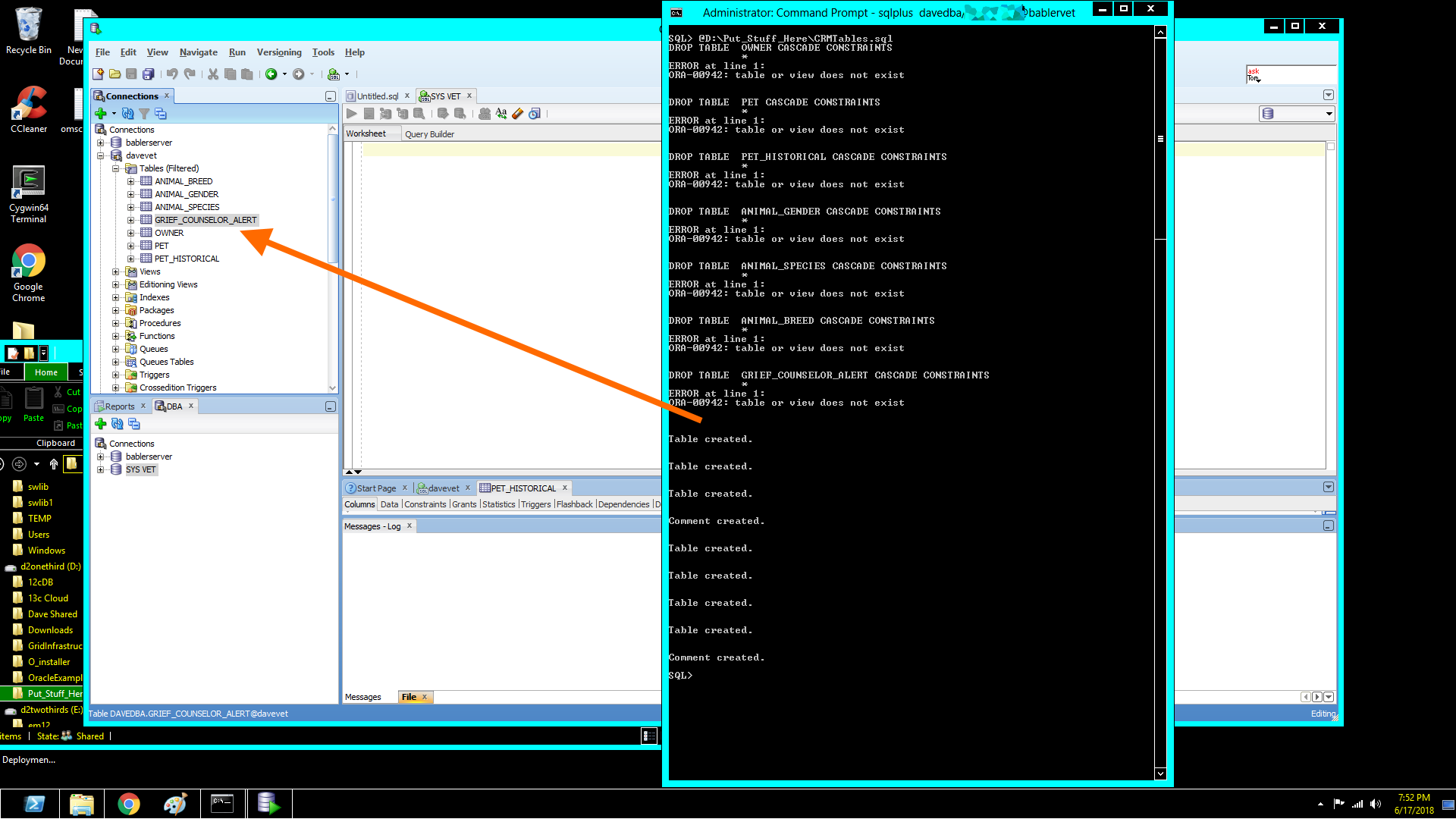
[](https://drive.google.com/open?id=1sPEWK_IDQ-Da93nKyHvWAB2Sc5xqvwjU)

Figure 9

Some tables with composite primary keys, and or unusual attributes were directly written into SQL+ or SQL Developer [figure 10].

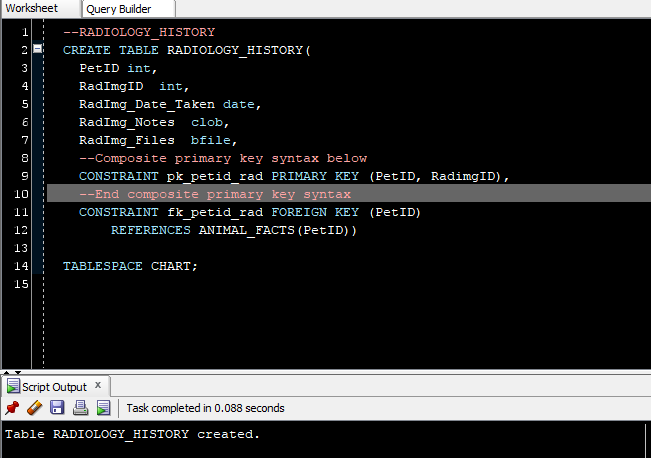
[](https://drive.google.com/open?id=1AEYkpo0_3yqFe4xFdT8scMdnxN_OeCao)

Figure 10

Tables with relatively few attributes were created with the SQL Developer GUI [figures 11-13].

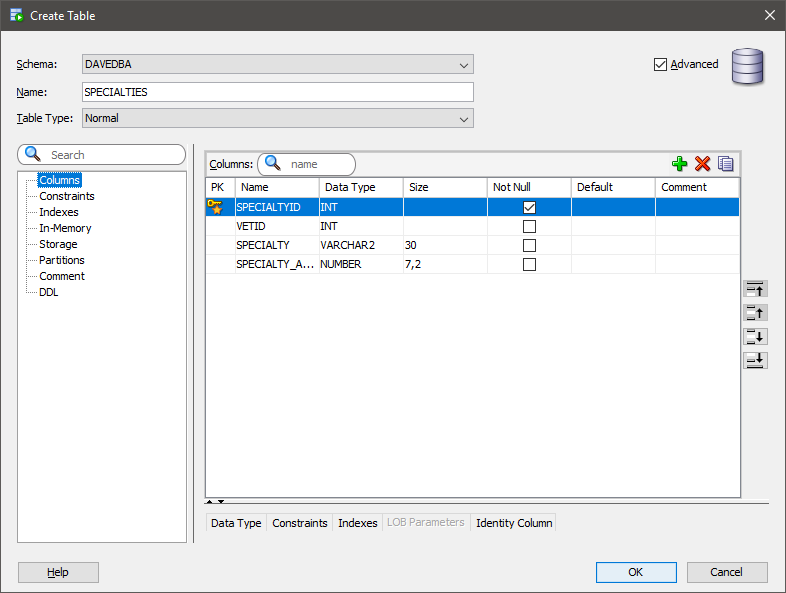
[](https://drive.google.com/open?id=1u8P7EEIXItnZzRSUYx9fZozEwPFWCIja)

Figure 11

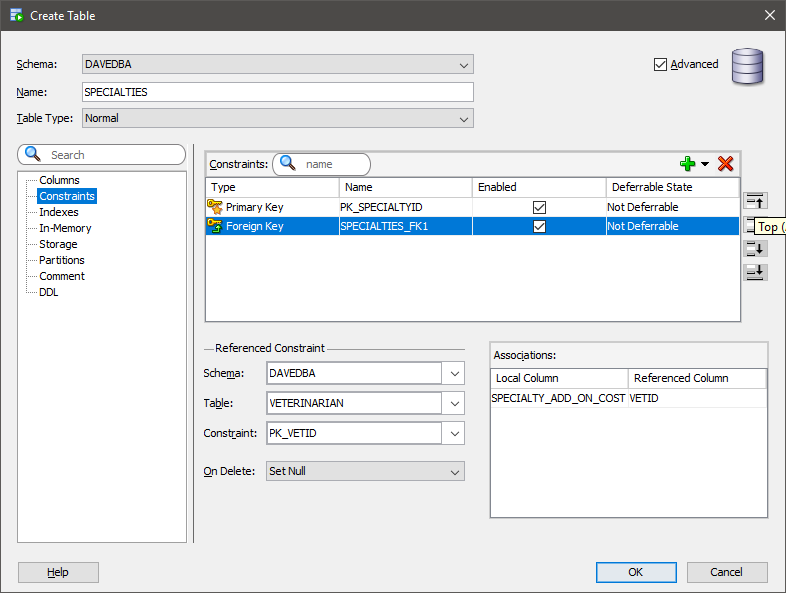
[](https://drive.google.com/open?id=1u8P7EEIXItnZzRSUYx9fZozEwPFWCIja)

Figure 12

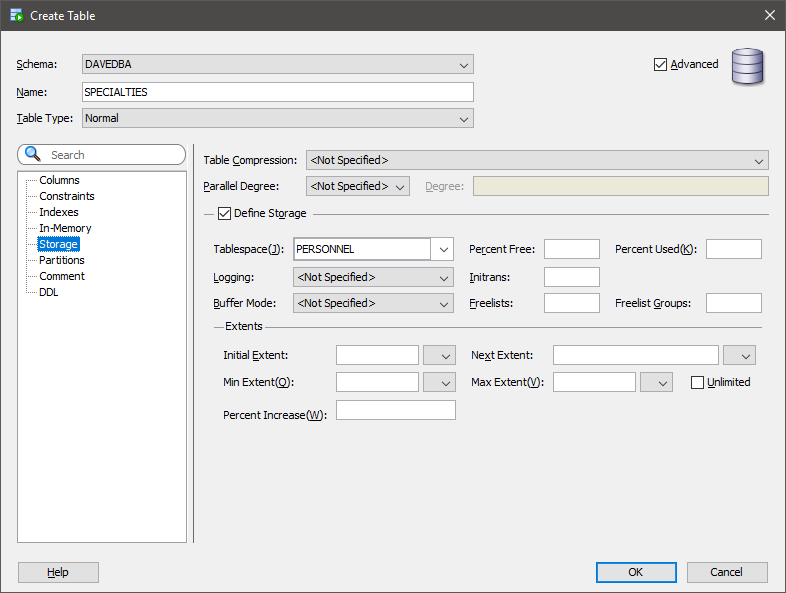
[](https://drive.google.com/open?id=1gJ6XMp0lwEhH-dYYUNej-DDq8JlsyrgU)

Figure 13

In certain instances, there were existing tables that needed to be altered after other tables were uploaded because of constraint violations and referential integrity concerns. These changes were typically handled with SQL+. The names of any tables proposed during requirements analysis that contained the term *Procedure* were given a prefix of *Vet\_* ; this was to avoid confusion with the Oracle structured program units called “Procedures” [figure 14].

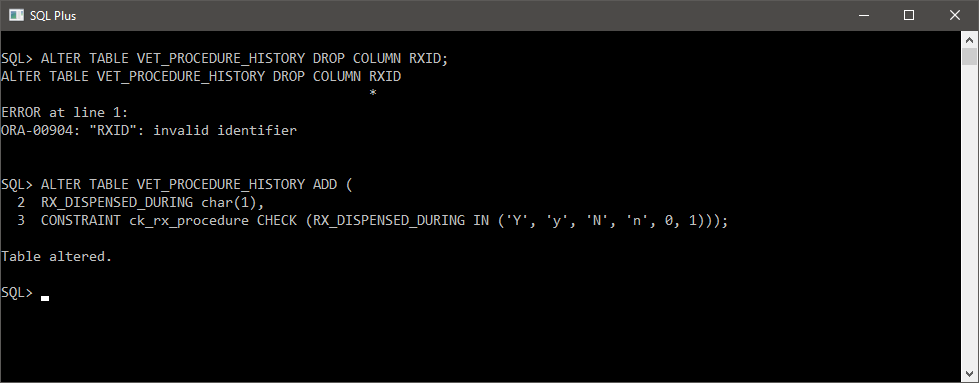


Figure 14

As work on the database progressed there were a few attributes that needed renaming, type conversions, or replacement in different tables. Figure 14 is an example of such an occurrence.

## Sequences and Identity Columns

Sequences and identity columns for Primary Keys and other unique identifiers described in CSFDAVD were added via SQL manually and by using the GUI functions of SQL Developer; all sequences and identity columns functioned as expected [figures 15 & 16].

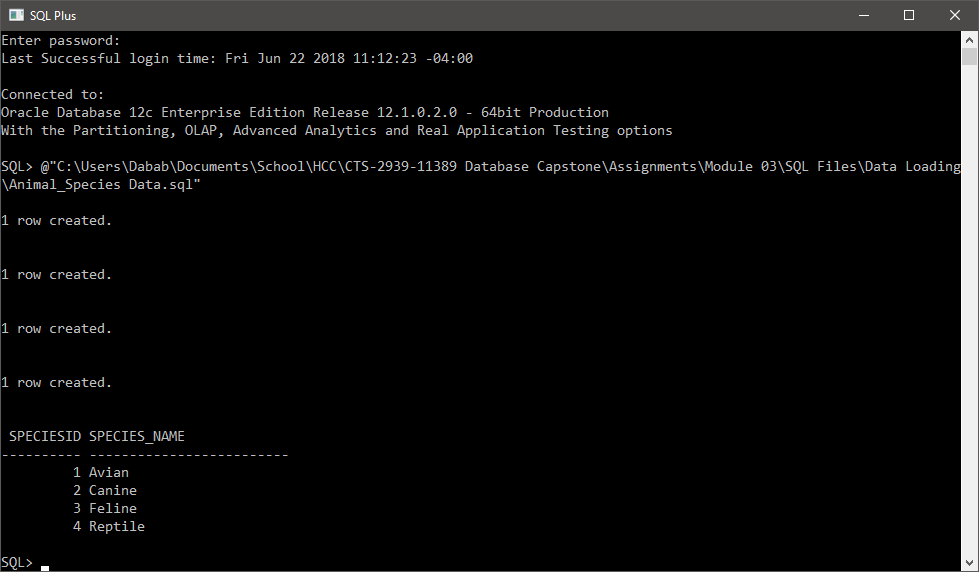
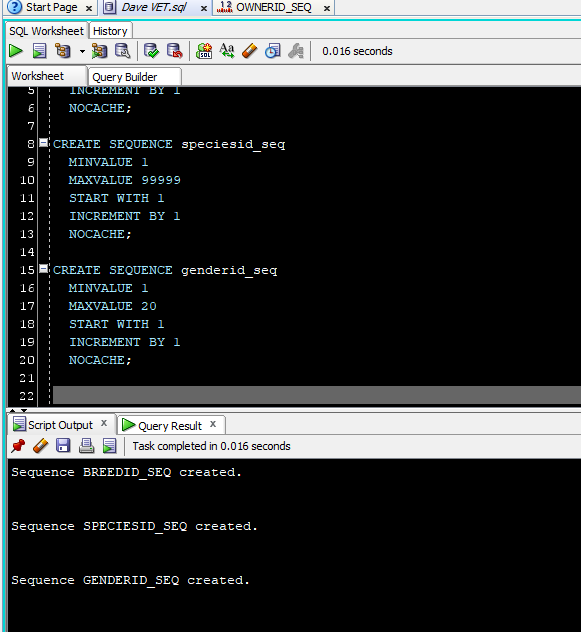


Figure 15

Figure 16

## Data Loading

Sample data for the database was synthesized using a combination of data from with online resources such as <http://generatedata.com> and <https://www.firstveterinarysupply.com> , along with fictitious data generated by the author based on past experience interacting with veterinary offices. Sample data was typically loaded in using SQL Developer’s efficient data loading program [figure 17].

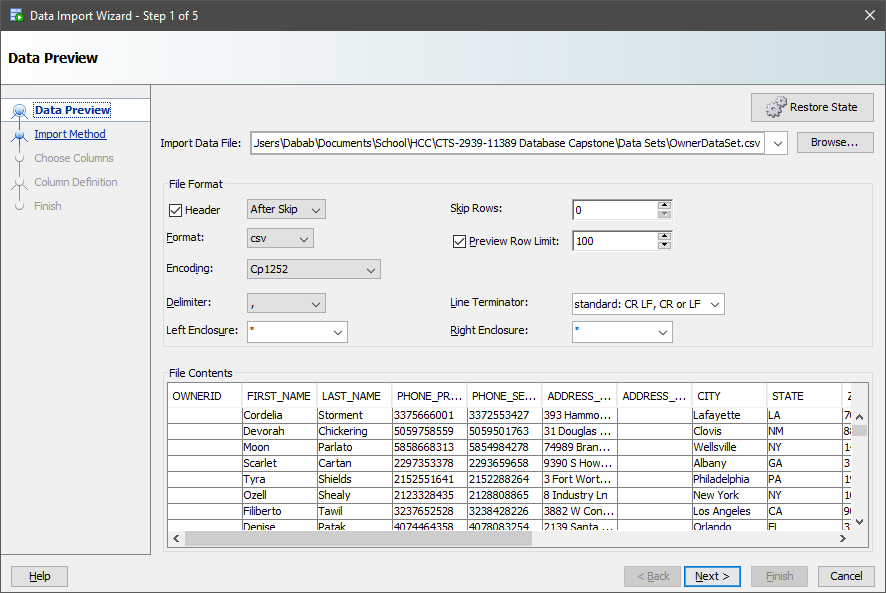


Figure 17

# BUSINESS RULES IMPLEMENTATION[[1]](#footnote-1)

## Record Keeping, Reception, Customer Relations Managment

### CRM-01 An Owner may have many pets; A pet may only belong to one owner.

### CRM-02[[2]](#footnote-2) An owner may designate a responsible 3rd party adult to retrieve the animal after clinical events.

These rules were straightforward to implement. The 3rd party adult was associated directly with the pet owner by adding those attributes to the owner table. A one-to-many [1:M] foreign key relationship was created between the Pet table and the Owner table [figure 18].

The creation of these tables also satisfied the following transaction requirements:

* TRX-07: Add new owner
* TRX-08: Add new pet



Figure 18

### CRM-07 List of all owners pets should be easily accessible

Figure 18 satisfies the basic intent of business rule CRM-07; adding the columns for the pet’s name, breed, species, and gender ID keys completes the needs to satisfy this rule. However, looking at a group of keys is not useful *information*. Rather than using several subqueries and JOINs to tie the data together, simple functions were created to show what species, breed, and gender an animal is based on an associated key [code insert 1 & figure 19].

**CREATE** **OR** **REPLACE** **FUNCTION** FUNC\_BREED(

f\_breedid **IN** ANIMAL\_BREED.BREEDID%**TYPE**)

**RETURN** varchar2

**AS**

lv\_breed\_name ANIMAL\_BREED.BREED\_NAME%**TYPE**;

lv\_except varchar2(100) 'No Breed Found';

**BEGIN**

**SELECT** BREED\_NAME

**INTO** lv\_breed\_name

**FROM** ANIMAL\_BREED

**WHERE** BREEDID = f\_breedid;

**RETURN** lv\_breed\_name;

**EXCEPTION**

**WHEN** NO\_DATA\_FOUND **THEN**

DBMS\_OUTPUT.PUT\_LINE('This breed id does not exist' ||chr(10)||

'Are you certain it has been typed in correctly?' );

**RETURN** **NULL**;

**END**;

Code Insert 01

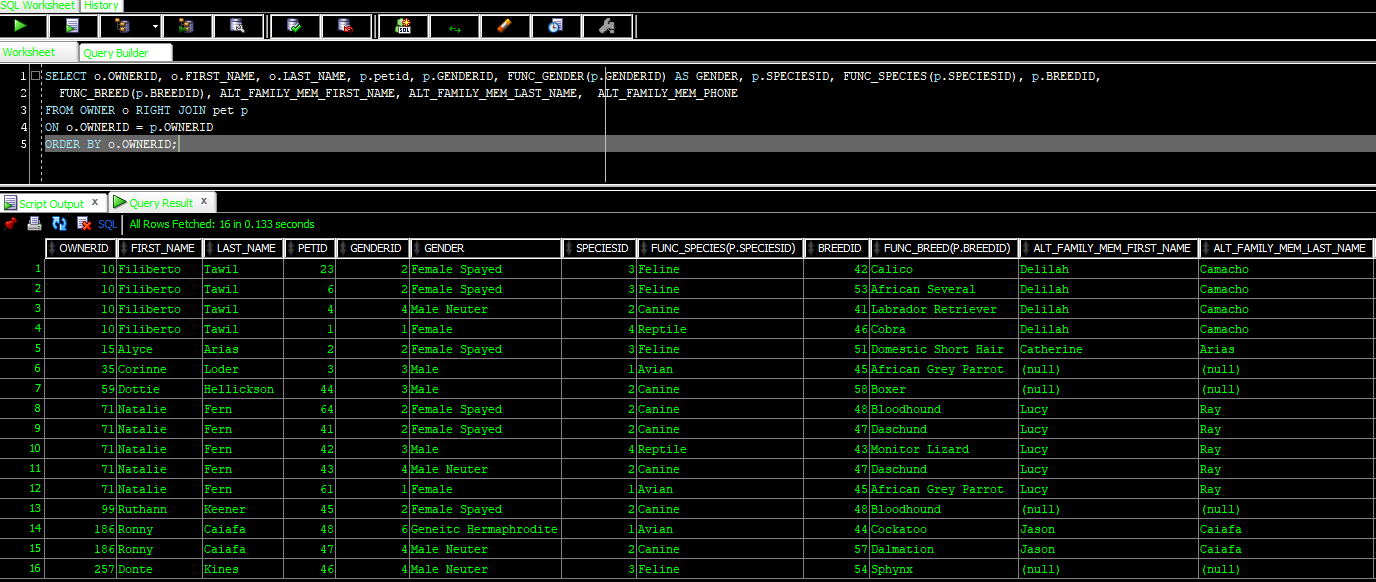
[](https://drive.google.com/open?id=1wfWOHECHH6e4olwVdgrAVC_ua5etJLE0)

Figure 19

### CRM-03 Show only living animals when an owner checks in for an appointment

Attempting to show just living pets using SQL alone would have involved at least two self-joins of the same table. This would lead to duplicate values and other anomalies. To prevent, this a function was developed that creates a comma-separated list inside of a column [code insert 02].

**CREATE** **OR** **REPLACE** **FUNCTION** FUNC\_PET\_SIBLINGS ( f\_petid **IN** int)

**RETURN** varchar2

**AS**

*--shows only living pets associated with an OWNERID*

lv\_ownerid int;

lv\_sibling\_name varchar2(500) := **NULL**;

lv\_isalive PET.IS\_LIVING%**TYPE**;

lv\_loop int :=0; *--we will use this for formatting text, because we are not trashy!*

**CURSOR** cur\_siblings **IS**

**SELECT** PET\_FIRST\_NAME

**FROM** PET

**WHERE** IS\_LIVING **IN** ('y', 'Y', '1') *--get living related to pets*

**AND** PETID <> f\_petid *--but not the same pet*

**AND** OWNERID = lv\_ownerid; *--get the Owner of the pets*

**BEGIN**

lv\_loop := 0;

DBMS\_OUTPUT.PUT\_LINE(lv\_loop);

**SELECT** OWNERID

**INTO** lv\_ownerid

**FROM** PET

**WHERE** PETID = f\_petid;

**FOR** rec\_sibling **IN** cur\_siblings LOOP

**CASE** *--while there is more pet names format the text and add it to the variable*

**WHEN** rec\_sibling.PET\_FIRST\_NAME **IS** **NOT** **NULL** **AND** lv\_loop >= 0

**THEN** lv\_sibling\_name := lv\_sibling\_name || rec\_sibling.PET\_FIRST\_NAME ||', ' ;

**ELSE** lv\_sibling\_name := lv\_sibling\_name ||'. ';

**END** **CASE**;

lv\_loop:= lv\_loop + 1;

**END** LOOP;

**RETURN** lv\_sibling\_name;

**END**;

Code Insert 02

This function was then used inside a Materialized View along with all other relevant data needed to check in a patient. When an owner comes in with their pets, the pets are usually either ill or anxious; thus, anything to make the check in practice quick will be appreciated from a customer stand point. To assist with this business, need a Materialized View was chosen instead of an ad-hoc query or a traditional View. To further speed up the process of patient check-in, non-unique indexes were built on the attributes of the pet’s first name and the owner’s last name; these are the columns most likely to be used to search for an incoming patient. The Materialized View refreshes only once an hour, as it is highly unlikely that a customer is going to have a dramatic change in data within an hour; and allows for reception to pull up the most accurate information as new customers come in for their appointments [figure 20] [[3]](#footnote-3).

[](https://drive.google.com/open?id=1oC0pItHwo180ZaH_TwweXTPhMPPjvEIK)

Figure 20

The combination of this code along with the code that satisfied [CRM-07](#_CRM-07_List_of) satisfied the reporting requirement: REPORT-05 CRM regarding reception and pet check-in.

### CRM-06 Grief counselor needs to know types of pet owner prefers

### CRM-04 Deceased pets should be placed in a historical table.

This was accomplished by tweaking the logic of the function that gives us an owner’s list of living pets. The modifications were:

* Removed the filter for living pets only (grief counselor wants to know what types of pet the owner has for the purpose of adoption suggestions if appropriate).
* Added a counter for each species.
* Added attributes for SpeciesID and BreedID [code insert 3].

**CREATE** **OR** **REPLACE** **FUNCTION** FUNC\_ALL\_OTHER\_PETS(

f\_petid **IN** int)

**RETURN** **clob**

**AS**

lv\_ownerid **OWNER**.OWNERID%**TYPE**;

lv\_sibling\_name varchar2(500) := **NULL**;

lv\_sibling\_species varchar2(500) :=**NULL**;

lv\_sibling\_breed varchar2(500) :=**NULL**;

*--using clobs not varchar because we have no clue how many animals someone will have over a lifetime*

lv\_other\_pets **clob** :=**NULL**; *--will be the builder each data type will get concated in there.*

lv\_avian **clob** := **NULL**;

lv\_canine **clob** := **NULL**;

lv\_feline **clob** := **NULL**;

lv\_reptile **clob** := **NULL**;

lv\_aviancount int :=0;

lv\_caninecount int := 0;

lv\_felinecount int := 0;

lv\_reptilecount int :=0;

lv\_loop int :=0;

*--the goal is to get ALL pets even previously dead ones so the counselor knows what the parent prefers in a pet*

**CURSOR** cur\_otherpets **IS**

**SELECT** s.speciesID, ab.BREED\_NAME, ab.breedid, s.SPECIES\_NAME, COLORING, PETID, OWNERID

**FROM** ANIMAL\_BREED ab **JOIN** ANIMAL\_SPECIES s

**ON** ab.SPECIESID = s.SPECIESID

**JOIN** PET p **ON** ab.BREEDID = p.BREEDID

**WHERE** ab.BREEDID

= **ANY** (**SELECT** BREEDID

**FROM** PET

**WHERE** PETID <> f\_petid

**AND** OWNERID = lv\_ownerid)

**AND** OWNERID = lv\_ownerid; *--have to re-restrict the ownerid because otherwise the subquery starts pulling unrelated pets based on species.*

**BEGIN**

**SELECT** OWNERID

**INTO** lv\_ownerid

**FROM** PET

**WHERE** PETID = f\_petid;

*--get the owner id from the dead pet, then loop all known pets out of the cursor*

*--and into a text format report*

**FOR** rec\_otherpets **IN** cur\_otherpets LOOP

**CASE**

**WHEN** rec\_otherpets.speciesID = 1 **THEN**

lv\_aviancount := lv\_aviancount + 1;

lv\_avian:= lv\_avian || rec\_otherpets.BREED\_NAME|| '-'|| rec\_otherpets.COLORING||' plumage' ||', ';

**WHEN** rec\_otherpets.speciesID = 2 **THEN**

lv\_caninecount := lv\_caninecount + 1;

lv\_canine:= lv\_canine || rec\_otherpets.BREED\_NAME|| '-'|| rec\_otherpets.COLORING||' fur' ||', ';

**WHEN** rec\_otherpets.speciesID = 3 **THEN**

lv\_felinecount := lv\_felinecount + 1;

lv\_feline:= lv\_feline || rec\_otherpets.BREED\_NAME|| '-'|| rec\_otherpets.COLORING||' fur' ||', ';

**WHEN** rec\_otherpets.speciesID = 4 **THEN**

lv\_reptilecount := lv\_reptilecount + 1;

lv\_reptile:= lv\_reptile || rec\_otherpets.BREED\_NAME|| '-'|| rec\_otherpets.COLORING||' scales' ||', ';

**ELSE** **NULL**;

**END** **CASE**;

lv\_loop := lv\_loop + 1;

**END** LOOP;

**CASE**

**WHEN** lv\_loop **IS** **NULL** **OR** lv\_loop = 0 **THEN**

lv\_other\_pets := 'No other (known) pets owned by this owner currently.';

**ELSE**

lv\_other\_pets := 'This animal has pet-siblings with the following species/breeds:' ||' '|| CHR(10);

IF lv\_aviancount > 0

**THEN** lv\_other\_pets := lv\_other\_pets || 'They own (or have owned) '||lv\_aviancount||' birds of the following breeds: '||lv\_avian||' '|| CHR(10);

**END** IF;

IF lv\_caninecount > 0

**THEN** lv\_other\_pets := lv\_other\_pets || 'They own (or have owned) '||lv\_caninecount||' dogs of the following breeds: '||lv\_canine||' '|| CHR(10);

**END** IF;

IF lv\_felinecount > 0

**THEN** lv\_other\_pets := lv\_other\_pets || 'They own (or have owned) '||lv\_felinecount||' cats of the following breeds: '||lv\_feline||' '|| CHR(10);

**END** IF;

IF lv\_reptilecount > 0

**THEN** lv\_other\_pets := lv\_other\_pets || 'They own (or have owned) '||lv\_reptilecount||' reptiles of the following breeds: '||lv\_reptile||' '|| CHR(10);

**END** IF;

lv\_other\_pets := lv\_other\_pets||'End of other known pets.';

**END** **CASE**;

DBMS\_OUTPUT.PUT\_LINE(lv\_other\_pets);

**RETURN** lv\_other\_pets;

**END**;

Code Insert 03

A table was created with the base values data the function will use; along with data for a function that shows the information about the death of the animal. Data is only imported into this table upon an animal being moved to the historical table and is marked dead; other inserts on the historical would be for inapplicable situations (for example: owner and pets have moved). To satisfy these requirements I used two pieces of logic:

1. A procedure that imports data about the death of the animal and fills in the appropriate attributes in the tables; then, the procedure creates a historical record for the animal [figure 21].
2. A trigger that fires once an insert has happened upon the historical table [code insert 4].

[](https://drive.google.com/open?id=1g9aiXKcjmod-F2FkWbEojtgBD3-ktY6Y)

Figure 21

**CREATE** **OR** **REPLACE** **TRIGGER** GRIEF\_ALERT\_TRG

**AFTER** **INSERT** **ON** PET\_HISTORICAL

**FOR** **EACH** **ROW**

**WHEN** (**NEW**.IS\_LIVING **IN** ('N','0', 'n'))

*--other inserts are irrelevant to this those are for pets that have moved away*

**DECLARE**

*/\*will grab from a JOIN of data that comes from*

*CRM and CHART areas of the database to avoid having to*

*either use a Global Temp table and/to avoid having to deal with*

*Mutating tables\*/*

lv\_petid GRIEF\_COUNSELOR\_ALERT.PETID%**TYPE** := :**NEW**.PETID;

lv\_ownerid GRIEF\_COUNSELOR\_ALERT.OWNERID%**TYPE**;

lv\_deathdate date;

lv\_phone **OWNER**.PHONE\_PRIMARY%**TYPE**;

**BEGIN**

**SELECT** o.OWNERID, o.PHONE\_PRIMARY, DEATH\_DATE

**INTO** lv\_ownerid, lv\_phone, lv\_deathdate

**FROM** ANIMAL\_FACTS af **JOIN** PET p

**ON** af.PETID = p.PETID **JOIN** **OWNER** o

**ON** p.OWNERID = o.OWNERID

**WHERE** af.PETID = lv\_petid;

**INSERT** **INTO** GRIEF\_COUNSELOR\_ALERT (ALERT\_DATE, PETID, OWNERID, PHONE\_PRIMARY, DEATH\_DATE)

**VALUES**(SYSDATE, lv\_petid, lv\_ownerid, lv\_phone, lv\_deathdate);

**END**;

Code Insert 04

Finally, a view was created which shows the grief counselor exactly what she needs to assist a grieving pet parent [figure 22].

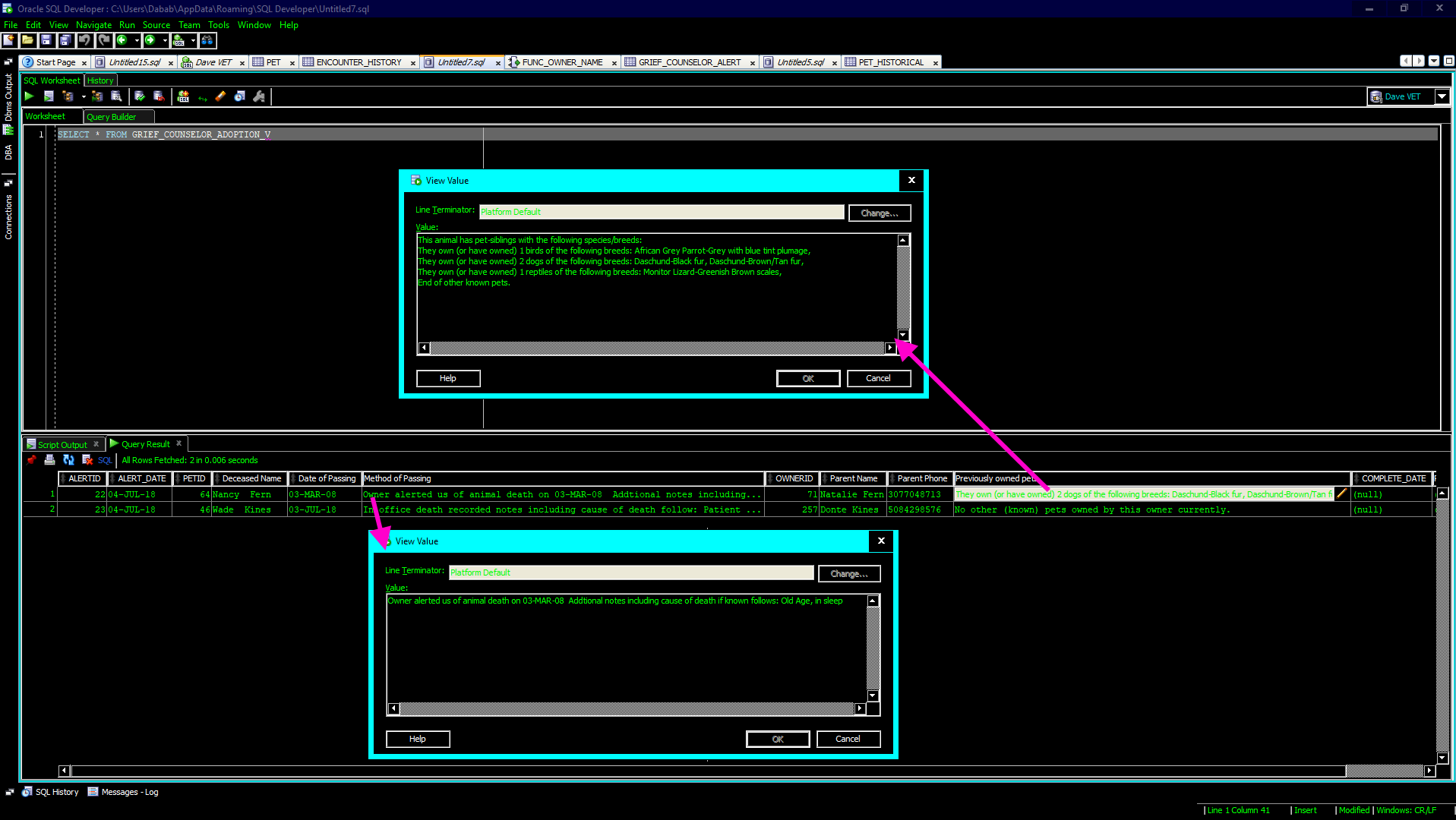
[](https://drive.google.com/open?id=1UKrtqVeD5quCuxUoD8atQ9aFPdtgjT1T)

Figure 22

### CRM-05 When a parent brings in a new pet a medical chart should be created

Per CSFDAVD, the medical chart starts with a table of basic information called Animal\_Facts; the reception/customer relations management table is simply called pet. When a new animal is added to the pet chart the trigger fires and the relevant data is automatically goes into the Animal\_Facts chart [figure 23]. This leads to data duplication in the database; duplicate data in this case is needed because it is not necessarily appropriate for the medical aspect of the patient’s data to be available to all employees of the business.

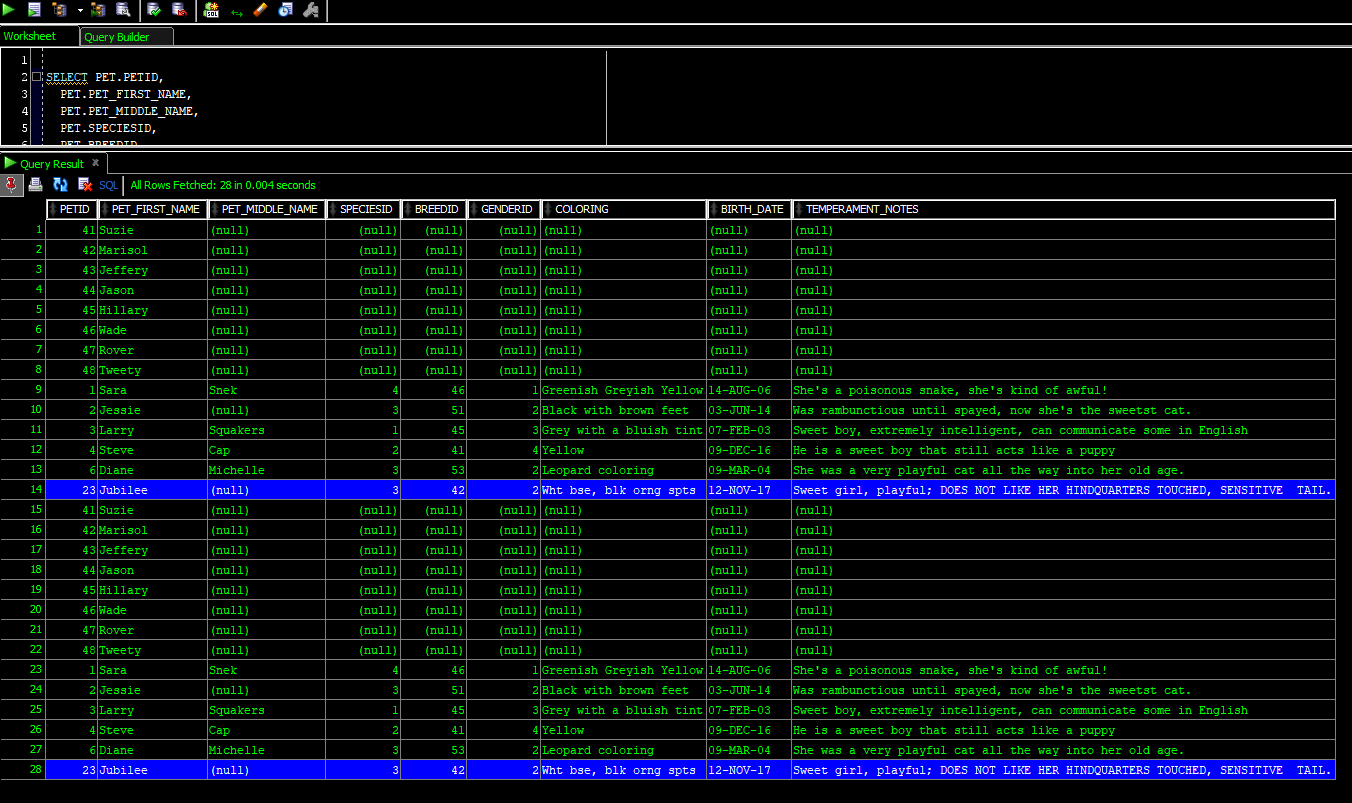
[](https://drive.google.com/open?id=1jXCEWZyuI98eHqWjvehtfuXDCxHWVvUu)

Figure 23

This code and data structure also satisfied the following transaction and reporting requirements:

* REPORT-06: Grief counseling—regarding the data needs for a grief counseling appointment.
* TRX-06: Pet historical.
* TRX-09: Update chart upon death

## Clinical Procedures

### PROC-01 Once a clinical event is complete it must be added to the chart

Clinical events, (AKA clinical procedures) range from complex surgeries to a simple patient checkup. Some procedures, like surgeries, involve the administering of several medications during the procedure. This means that data is potentially interfacing with three tables simultaneously, while other tables are waiting for a trigger to fire based on this data. To handle *clinical procedures,* I used an *Oracle Procedure*. This procedure takes advantage of Oracle’s ability to take in optional incoming parameters into the program unit, allowing for a simple clinical event with no medication or a complex surgery with up to five medications. If the vets feel that five medications is insufficient, the number of parameters can be easily expanded by copying code and modifying variable numbers up to PL/SQL’s maximum of 32768 incoming program unit variables (“Database PL/SQL Language Reference,” n.d.) . Each medical administration would just be one dose at a time, as a doctor would never say “give two doses of 30 cc Anetrizine” she would simply say “push 60 cc Anetrizine stat”. Per business rule RX-09: doctors and Vets do not write formal prescriptions for medicines given during a clinical event, this was also considered during the creation of this procedure. Finally, logic involving creating a way for these medicines to be documented, and marked as approved for regulatory reporting requirements, was added to logic and structures to fulfill all of the needs related to PROC-01 [code insert 05][[4]](#footnote-4).

**CREATE** **OR** **REPLACE** **PROCEDURE** PROC\_RX\_VETPROC(

p\_vetid **IN** int,

p\_petid **IN** int,

p\_vet\_procid **IN** int,

p\_proc\_notes **IN** **clob**,

p\_rxdisp **IN** char,

p\_drugid01 **IN** int **DEFAULT** **NULL**,

p\_drugdose01 **IN** varchar2 **DEFAULT** **NULL**,

p\_drug\_units01 **IN** number **DEFAULT** **NULL**,

p\_drugid02 **IN** int **DEFAULT** **NULL**,

p\_drugdose02 **IN** varchar2 **DEFAULT** **NULL**,

p\_drug\_units02 **IN** number **DEFAULT** **NULL**,

p\_drugid03 **IN** int **DEFAULT** **NULL**,

p\_drugdose03 **IN** varchar2 **DEFAULT** **NULL**,

p\_drug\_units03 **IN** number **DEFAULT** **NULL**,

p\_drugid04 **IN** int **DEFAULT** **NULL**,

p\_drugdose04 **IN** varchar2 **DEFAULT** **NULL**,

p\_drug\_units04 **IN** number **DEFAULT** **NULL**,

p\_drugid05 **IN** int **DEFAULT** **NULL**,

p\_drugdose05 **IN** varchar2 **DEFAULT** **NULL**,

p\_drug\_units05 **IN** number **DEFAULT** **NULL**

)

**AS**

*/\*This is for entering simple procedures with up to 5 medicines administered during,*

*note the incoming parameters with default values of null. This creates an optional list of drugs that can be administered during a surgery;*

*Then at a later time they can run the report that shows exactly what was given to the animal and update tables accordingly\*/*

lv\_ppprocid int;

lv\_rxnotes **clob**;

*/\*rotating one lv\_rxid variable and reinitializing it to save on RAM\*/*

lv\_rxid int;

**BEGIN**

**INSERT** **INTO** VET\_PROCEDURE\_HISTORY (VETID, PETID, VET\_PROCEDUREID, VET\_PROCEDURE\_NOTES, VET\_PROCEDURE\_DATE, RX\_DISPENSED\_DURING)

**VALUES**(p\_vetid, p\_petid, p\_vet\_procid, p\_proc\_notes, SYSDATE, p\_rxdisp)

RETURNING PATIENT\_VET\_PROCEDUREID **INTO** lv\_ppprocid;

**COMMIT**;

lv\_rxid :=0;

IF p\_drugid01 **IS** **NOT** **NULL**

**THEN** **INSERT** **INTO** RX\_HISTORY(VETID, PETID, DRUGID, DRUG\_DOSAGE, DATE\_WRITTEN, PATIENT\_VET\_PROCEDUREID, DRUG\_UNITS\_PRESCRIBED, DATE\_FILLED)

**VALUES**(p\_vetid, p\_petid, p\_drugid01, p\_drugdose01, SYSDATE, lv\_ppprocid, p\_drug\_units01, SYSTIMESTAMP)

RETURNING RXID **INTO** lv\_rxid;

**COMMIT**;

**INSERT** **INTO** RX\_ORDER(RXID, VETID, PETID, DRUGID, DRUG\_DOSAGE, DATE\_SUBMITTED,VET\_PROCEDUREID, CONTROLLED\_CHECKER, DRUG\_UNITS\_PRESCRIBED)

**VALUES**(lv\_rxid, p\_vetid, p\_petid, p\_drugid01, p\_drugdose01, SYSDATE, p\_vet\_procid, 0, p\_drug\_units01);

*/\*A NOTE ON CONTROL\_CHECKER according to the vets anytime a dose is given during an operation it will always be one dose, but is being set to a control check of 0*

*to distinguish it from proper RXs \*/*

**COMMIT**;

**END** IF;

IF p\_drugid02 **IS** **NOT** **NULL**

**THEN** **INSERT** **INTO** RX\_HISTORY(VETID, PETID, DRUGID, DRUG\_DOSAGE, DATE\_WRITTEN, PATIENT\_VET\_PROCEDUREID, DRUG\_UNITS\_PRESCRIBED)

**VALUES**(p\_vetid, p\_petid, p\_drugid02, p\_drugdose02, SYSDATE, lv\_ppprocid, p\_drug\_units02)

RETURNING RXID **INTO** lv\_rxid;

**COMMIT**;

**INSERT** **INTO** RX\_ORDER(RXID, VETID, PETID, DRUGID, DRUG\_DOSAGE, DATE\_SUBMITTED,VET\_PROCEDUREID, CONTROLLED\_CHECKER, DRUG\_UNITS\_PRESCRIBED, DATE\_FILLED)

**VALUES**(lv\_rxid, p\_vetid, p\_petid, p\_drugid02, p\_drugdose02, SYSDATE, p\_vet\_procid, 0, p\_drug\_units02, SYSTIMESTAMP);

**COMMIT**;

**END** IF;

IF p\_drugid03 **IS** **NOT** **NULL**

**THEN** **INSERT** **INTO** RX\_HISTORY(VETID, PETID, DRUGID, DRUG\_DOSAGE, DATE\_WRITTEN, PATIENT\_VET\_PROCEDUREID, DRUG\_UNITS\_PRESCRIBED)

**VALUES**(p\_vetid, p\_petid, p\_drugid03, p\_drugdose03, SYSDATE, lv\_ppprocid, p\_drug\_units03)

RETURNING RXID **INTO** lv\_rxid;

**COMMIT**;

**INSERT** **INTO** RX\_ORDER(RXID, VETID, PETID, DRUGID, DRUG\_DOSAGE, DATE\_SUBMITTED,VET\_PROCEDUREID, CONTROLLED\_CHECKER, DRUG\_UNITS\_PRESCRIBED, DATE\_FILLED )

**VALUES**(lv\_rxid, p\_vetid, p\_petid, p\_drugid03, p\_drugdose03, SYSDATE, p\_vet\_procid, 0, p\_drug\_units03, SYSTIMESTAMP);

**COMMIT**;

**END** IF;

IF p\_drugid04 **IS** **NOT** **NULL**

**THEN** **INSERT** **INTO** RX\_HISTORY(VETID, PETID, DRUGID, DRUG\_DOSAGE, DATE\_WRITTEN, PATIENT\_VET\_PROCEDUREID, DRUG\_UNITS\_PRESCRIBED)

**VALUES**(p\_vetid, p\_petid, p\_drugid04, p\_drugdose04, SYSDATE, lv\_ppprocid, p\_drug\_units04)

RETURNING RXID **INTO** lv\_rxid;

**COMMIT**;

**INSERT** **INTO** RX\_ORDER(RXID, VETID, PETID, DRUGID, DRUG\_DOSAGE, DATE\_SUBMITTED,VET\_PROCEDUREID, CONTROLLED\_CHECKER, DRUG\_UNITS\_PRESCRIBED, DATE\_FILLED )

**VALUES**(lv\_rxid, p\_vetid, p\_petid, p\_drugid04, p\_drugdose04, SYSDATE, p\_vet\_procid, 0, p\_drug\_units04, SYSTIMESTAMP);

**COMMIT**;

**END** IF;

IF p\_drugid05 **IS** **NOT** **NULL**

**THEN** **INSERT** **INTO** RX\_HISTORY(VETID, PETID, DRUGID, DRUG\_DOSAGE, DATE\_WRITTEN, PATIENT\_VET\_PROCEDUREID, DRUG\_UNITS\_PRESCRIBED)

**VALUES**(p\_vetid, p\_petid, p\_drugid05, p\_drugdose05, SYSDATE, lv\_ppprocid, p\_drug\_units05)

RETURNING RXID **INTO** lv\_rxid;

**COMMIT**;

**INSERT** **INTO** RX\_ORDER(RXID, VETID, PETID, DRUGID, DRUG\_DOSAGE, DATE\_SUBMITTED,VET\_PROCEDUREID, CONTROLLED\_CHECKER, DRUG\_UNITS\_PRESCRIBED, DATE\_FILLED )

**VALUES**(lv\_rxid, p\_vetid, p\_petid, p\_drugid05, p\_drugdose05, SYSDATE, p\_vet\_procid, 0, p\_drug\_units05, SYSTIMESTAMP);

**COMMIT**;

**END** IF;

**END**;

Code Insert 05

Figure 24 shows the successful[[5]](#footnote-5) running of the procedure along the raw data that allows pharmacology, or another vet, to verify and approve any medicines dispensed during a clinical event. Figure 25 shows the simplified view that would be shown to an actual person. The following figure shows the updates to all of tables that this program unit affects including VET\_PROCEDURE\_HISTORY[[6]](#footnote-6). Note that there is no staff attribute, this is to satisfy business rule [BR] CHART-17, that all procedures must be done under the supervision of a fully accredited veterinary doctor.

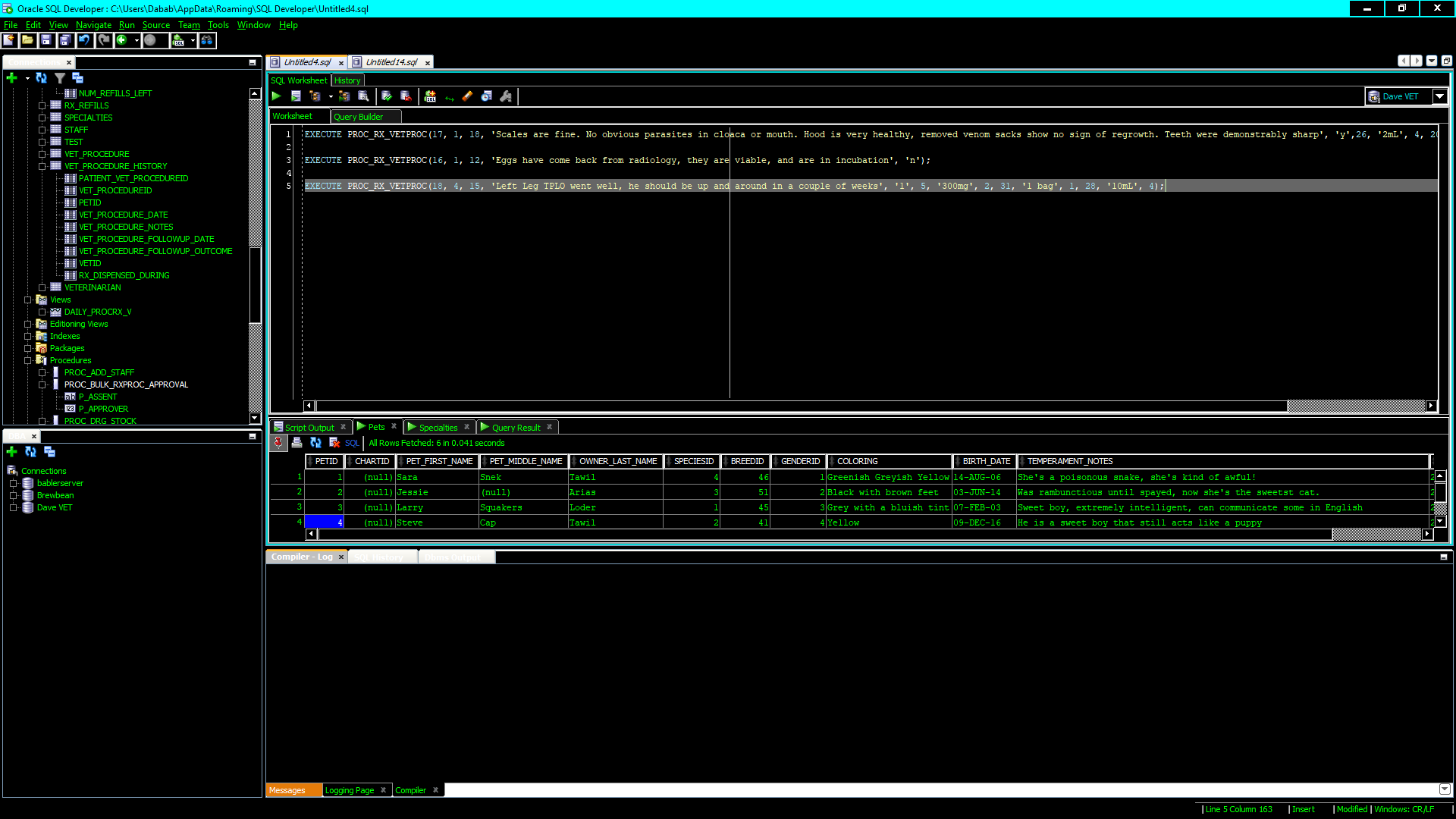
[](https://drive.google.com/open?id=1E6ma2ZdU9wVD1_K-XKPg9Fip5X5jAWEl)

Figure 24

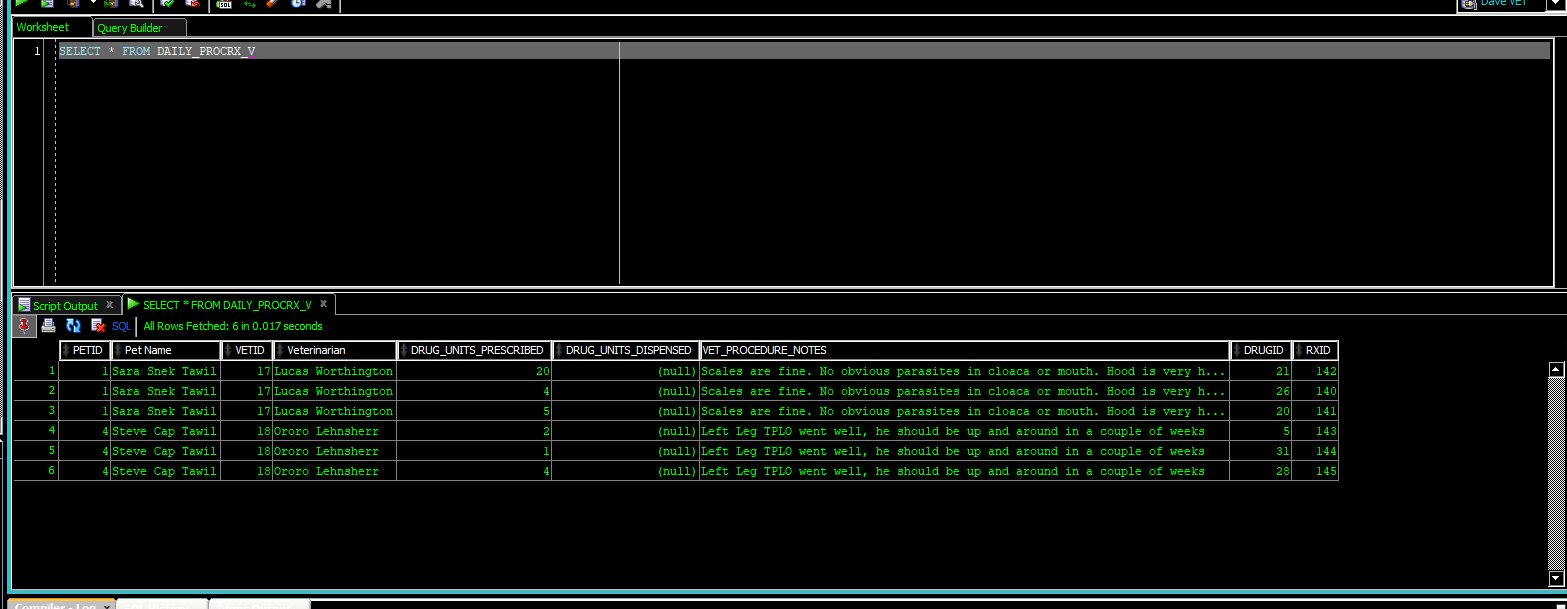
[](https://drive.google.com/open?id=1LQ6gLjQb4HV0lbF7BZfSUNTk10Sro3AD)

Figure 25

[](https://drive.google.com/open?id=1ikB-wjOBgInWBb0930kTNjhwshMlp_A7)

Figure 26

A procedure was created to bulk approve the clinical event generated pseudo-prescriptions in one command [figure 27]; once approved that medicine is removed from the view [figure 28]. Finally all appropriate tables are updated [figure 29].

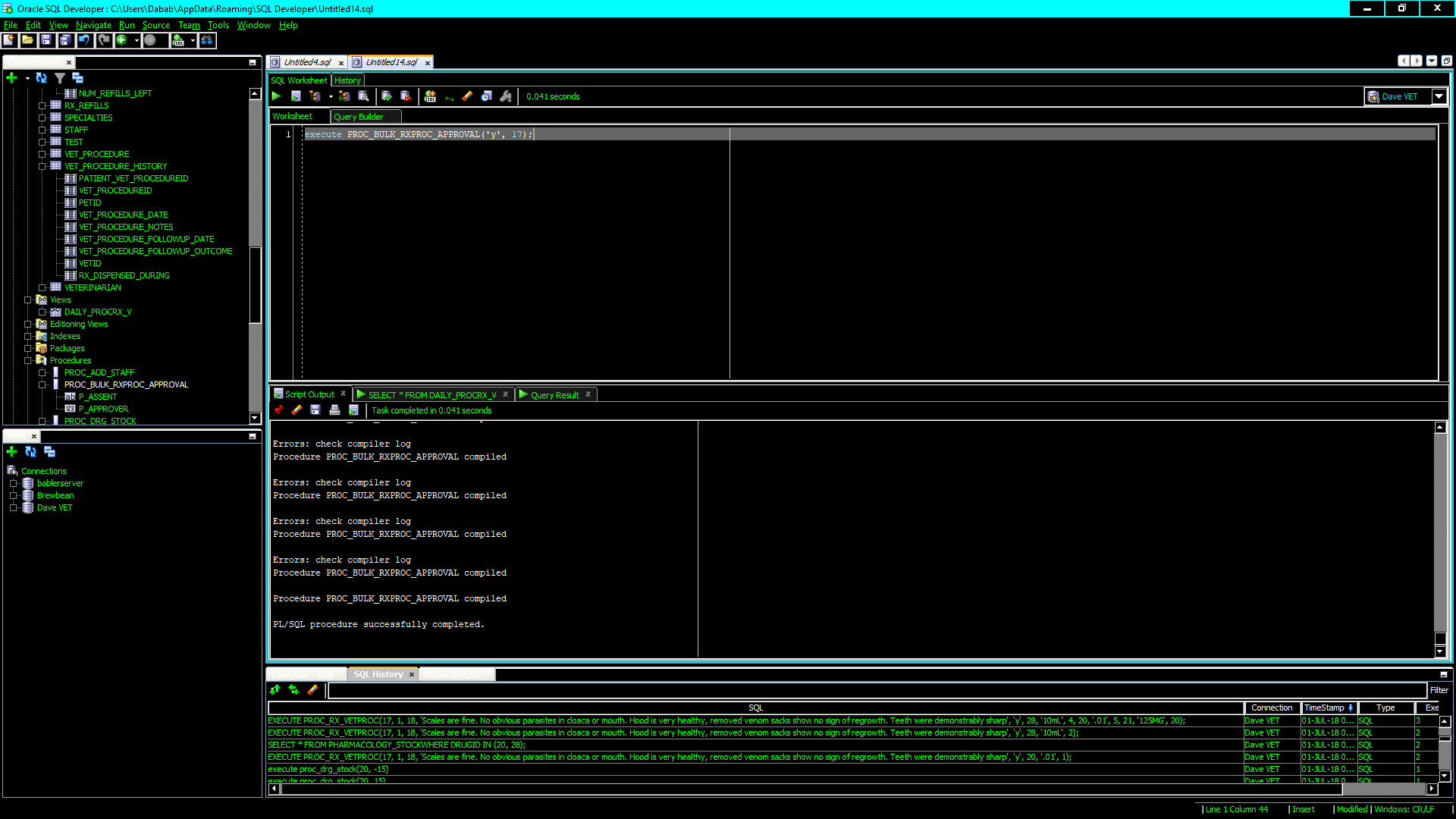
[](https://drive.google.com/open?id=1x5KmKdCKcbbjrN6ti9pIj29_bOx5mdkJ)

Figure 27

[](https://drive.google.com/open?id=17Ck4Z5gsGXinlc7MkRLeU3tNbQMHq0ZJ)

Figure 28

[](https://drive.google.com/open?id=1o49CJm3Z7xIvIpXKsPpNxAtvmJIJXRlT)

Figure 29

This also satisfied the following requirements:

* RX-09: Medicines given during surgery.
* CHART-17: All medical procedures done by non-vet staff must be supervised by a licensed vet.
* CHART-08: All medical procedures done on the animal must be stored in the chart.
* TRX-13: Veterinary procedures.

## Pharmacology, Pathology, Blood Bank

### RX-04 Add ℞ and Pathology Lab Orders to a View

Rather than having a complicated view with different types of work showing two views were created: one for incoming ℞ orders and another for incoming pathology tests [figures 30 & 31].

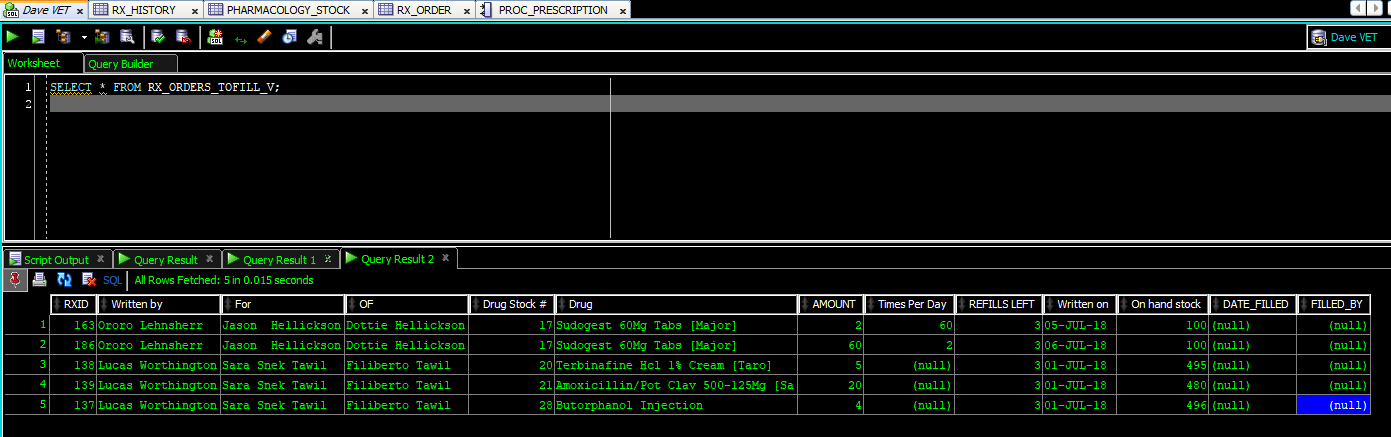
[](https://drive.google.com/open?id=1dK2p9JV6K2xlOmVojdRjAP8B4GcP2Hln)

Figure 30

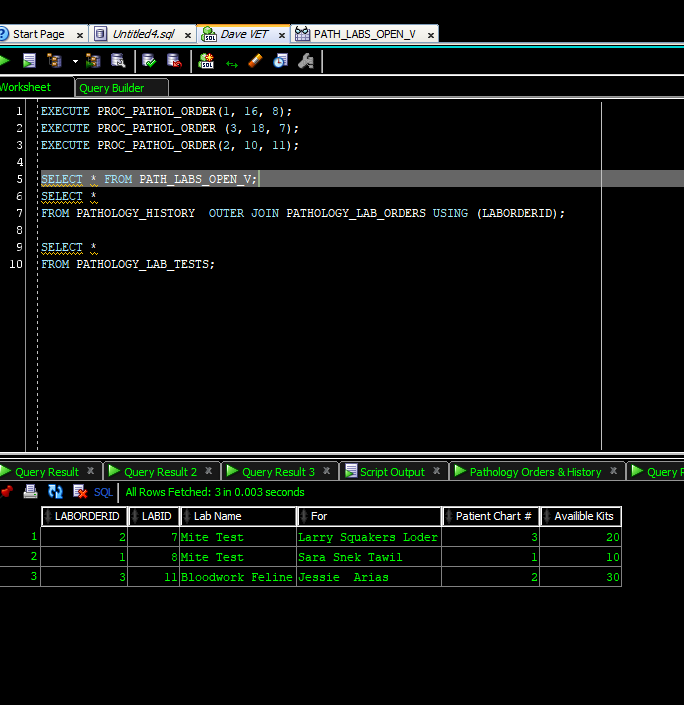
[](https://drive.google.com/open?id=1Y_t2gqY1l4fa_qOggc335uz6LWy7q-Ac)

Figure 31

To make these views work the following needed to be built first:

* A procedure for a vet to place a prescription [PROC\_PRESCRIPTION].
* A procedure for a vet to place a pathology lab order [PROC\_PATHOL\_ORDER] [figure 31].
* Functions that show verbose lab names, vet names, and chart names (humans don’t respond well to “Fill prescription for patient 10, from vet 15, using drug 98”) [various].
* A procedure to complete a pathology lab order [PROC\_PATH\_RESULTS] [figure 32].
* Folded into this program unit is a way of informing veterinarians if pathology has detected a critical illness (such as FIV, malignant tumors, avian-flu, etc.).
* Also folded into this logic is a way of subtracting lab-kits from inventor [figure 33].
* A procedure to fill a prescription [PROC\_RX\_FILL] that:
* Does the appropriate insertions/updates on the appropriate tables (RX\_HISTORY, RX\_REFILLS, etc.).
* Subtracts the drug units from inventory and prevents a ℞ from being filled if there is not enough in stock. Accomplished by using a sub-unit procedure [PROC\_DRG\_STOCK].
* Sends out exceptions, notes the account, and prevents transaction completion, if the RX\_ORDER table constraint CHK\_DISPENSE is violated (you cannot fill a ℞ for more medicine than what is prescribed).
* Sends out exceptions, notes the account, and prevents transaction completion, if the RX\_ORDER table constraint CHK\_CONTR is violated (no controlled substance ℞ can be filled for more than 14 days at a time).
* Sends out exceptions, notes the account, and prevents transaction completion if a vet accidentally submits the same prescription (drug, drug dose, times per day) twice in one day [figure 34]

Once this logic was built then the views shown previously in figures 30 & 31 could be created and tested.

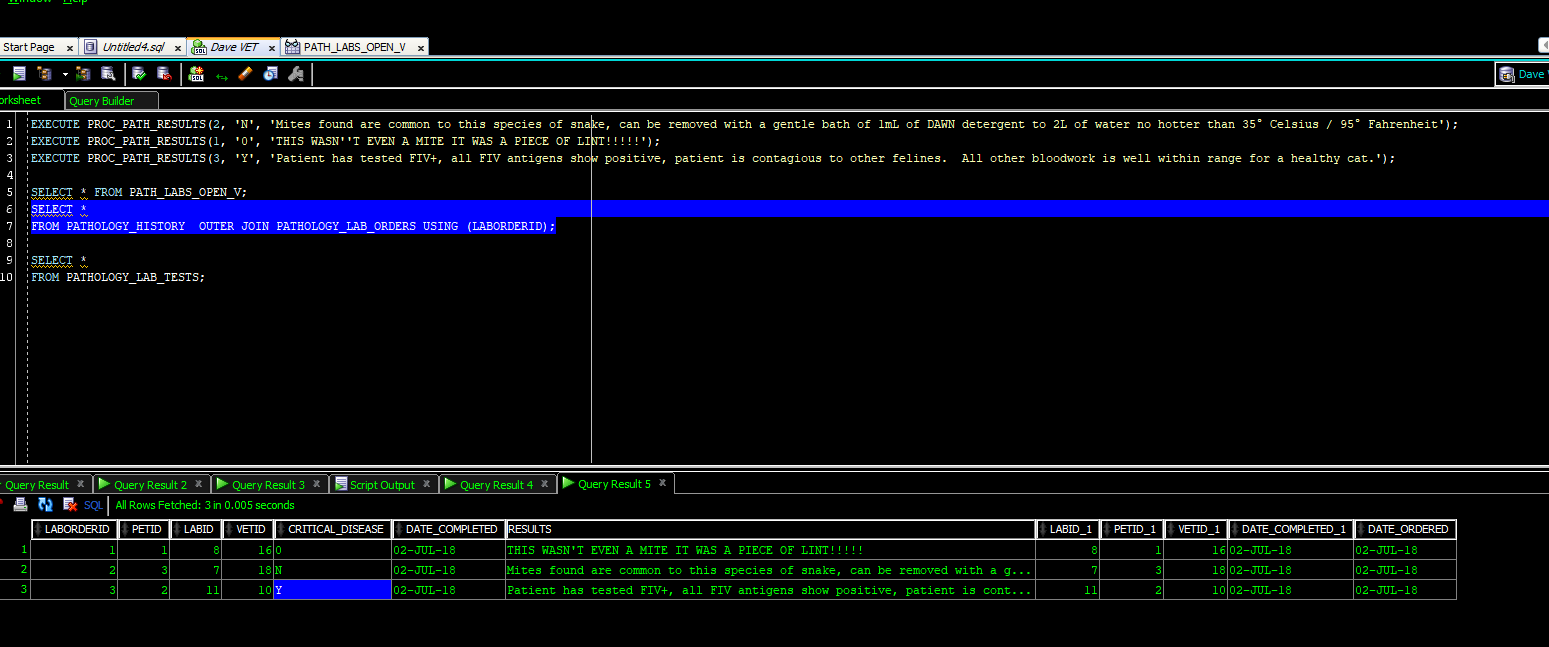
[](https://drive.google.com/open?id=1ny7EPd8O1Xge6vLZdnXpNHtZ6nOw8taz)

Figure 32

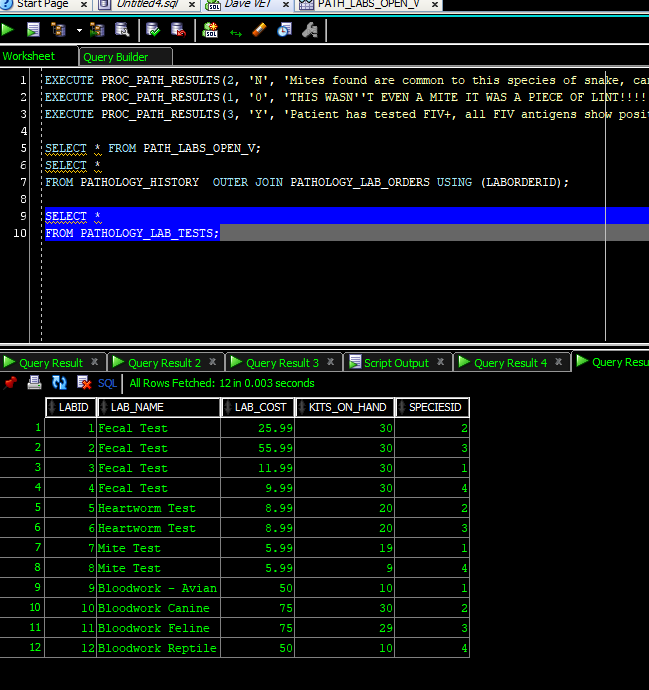


Figure 33

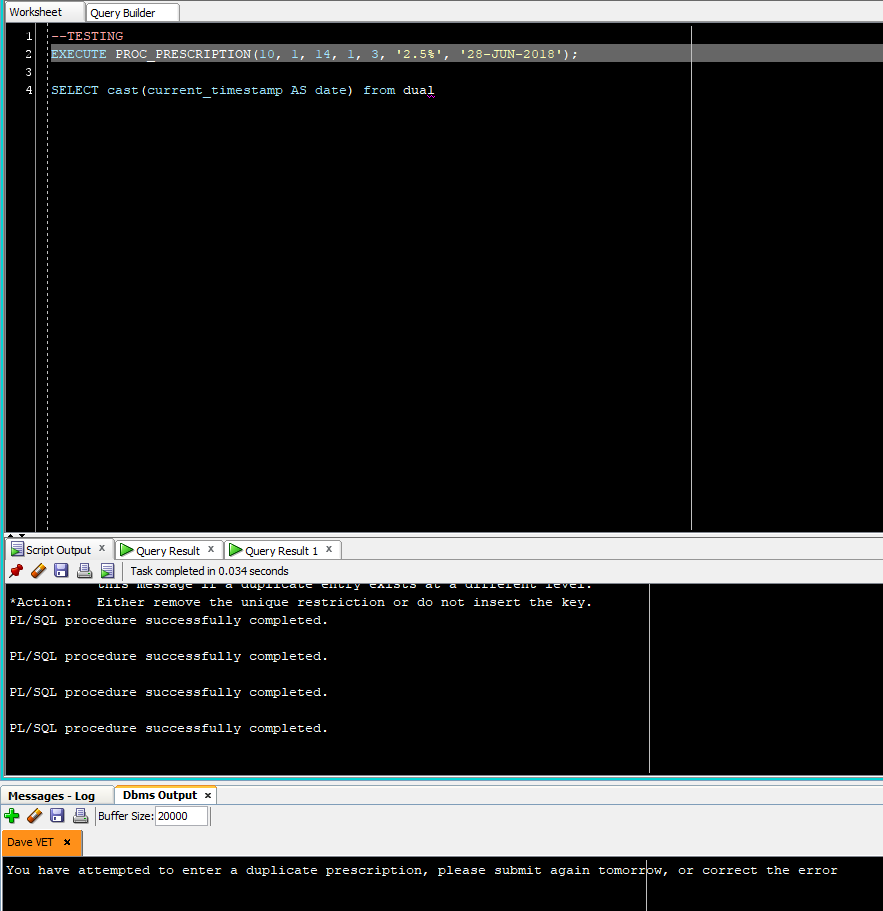


Figure 34

The logic from all these program units and data structures that satisfies B.R. RX-04 also fulfils the following:

* TRX-02 & RX-05: Subtract used lab kits from inventory.
* TRX-04: Update chart with pathology results.
* TRX-05 & CHART-11: Critical illness flag
* TRX-10 & RX-03: ℞ filled, inventory reduced, patient chart updated
* RX-01: Controlled substances limitations (no more than 14 days at a time).
* RX-03: Reduce medicine inventory upon successful filling of prescription.

## RX-10 a separate id for each individual prescription external to the chart

This is a government regulation explained to the author as: each prescription must have effectively two primary keys; one key relates to the patient’s chart and is used internally; one key is generated using a separate algorithm, unrelated to the patient’s chart. This is done incase all pharmacology tables need to be placed in a different database or data structure due to future regulation. This was accomplished by adding a separate attribute using Oracle’s Identity Column Function. Note the difference in figure 35 between the REGULATORYIDENTITYNUMBER and RXID.

[](https://drive.google.com/open?id=1hw8dIVL3MVAQWUjD5mNqSH090K-nYGca)

Figure 35

## RX-02 Stock reorder flag

This business rule was set to flag all drugs when the stock falls below 10 units. Additional requirements clarification was done with vets and the chemists and they decided that they would like to be able to set a reorder level based on the needs of the clinic per drug. This was accomplished by building an attribute for a Boolean field, and an integer field that would act as a floor; once the floor is reached a trigger will fire changing the flag thus alerting the staff it’s time to re-order. Oracle does not have a proper Boolean[[7]](#footnote-7) attribute data type, to ameliorate this problem a pseudo-Boolean column was created using Character (Char) datatype with a length of one. Instead of putting the onus on the application programming team to determine which specific characters would count as True or False options representing the most likely values were programmed into the check columns of {‘1’, ‘y’, ‘Y’} for true, and {‘0’, ‘n’, ‘N’} for false [figure 35]. Additional values can be added to the check constraints and the program units that call those columns should application-development request it.

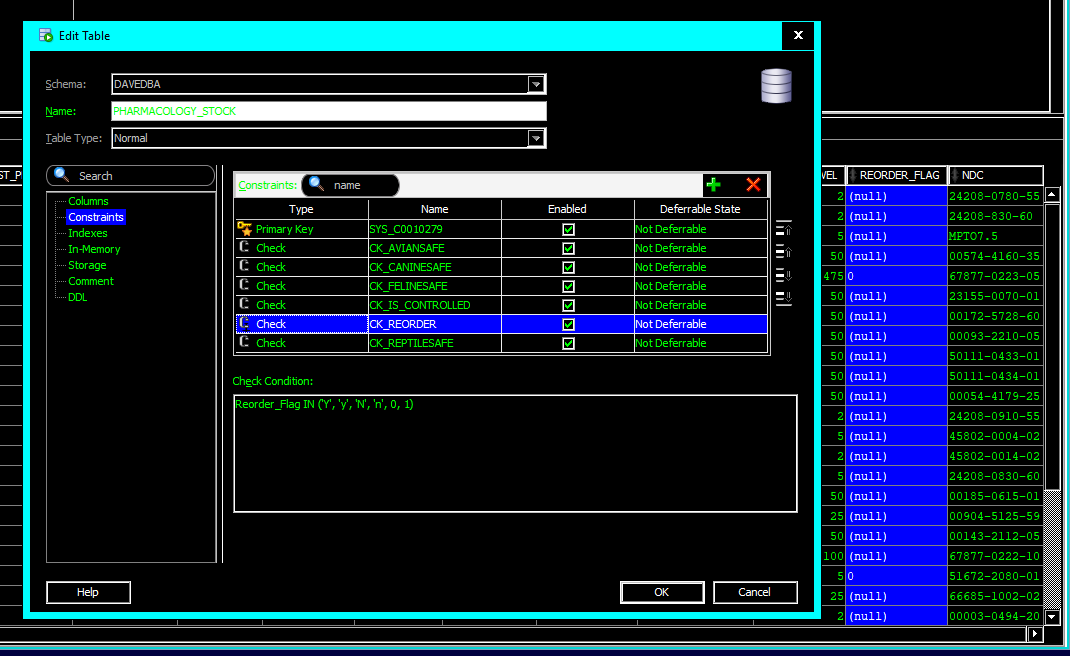
[](https://drive.google.com/open?id=1_fj_hVMh4iVTanRxW6s2cuXZ6wUVMUVZ)

Figure 36

Oracle also does not allow a trigger to read a table then alter that same table based on the new value and trying to get around this can create a phenomenon referred to as a mutating table (Casteel, 2013). To solve this problem, procedure program units were used. Once a prescription is filled that will bring the on-hand quantity equal to or less than the ORDER\_LEVEL attribute’s value, the procedure changes the flag. First a procedure PROC\_DRG\_STOCK was created to update the PHARMACOLOGY\_STOCK table based on the units filled for a prescription, and then change the flag column based on order level requirements. Then, that procedure was folded into both prescription filling/approval procedures PROC\_RX\_FILL and PROC\_BUL\_RX\_PROC\_APPROVAL. This allows for the stock table to be updated properly for both traditional prescription fills, and drugs dispensed during a clinical event [figure 37].

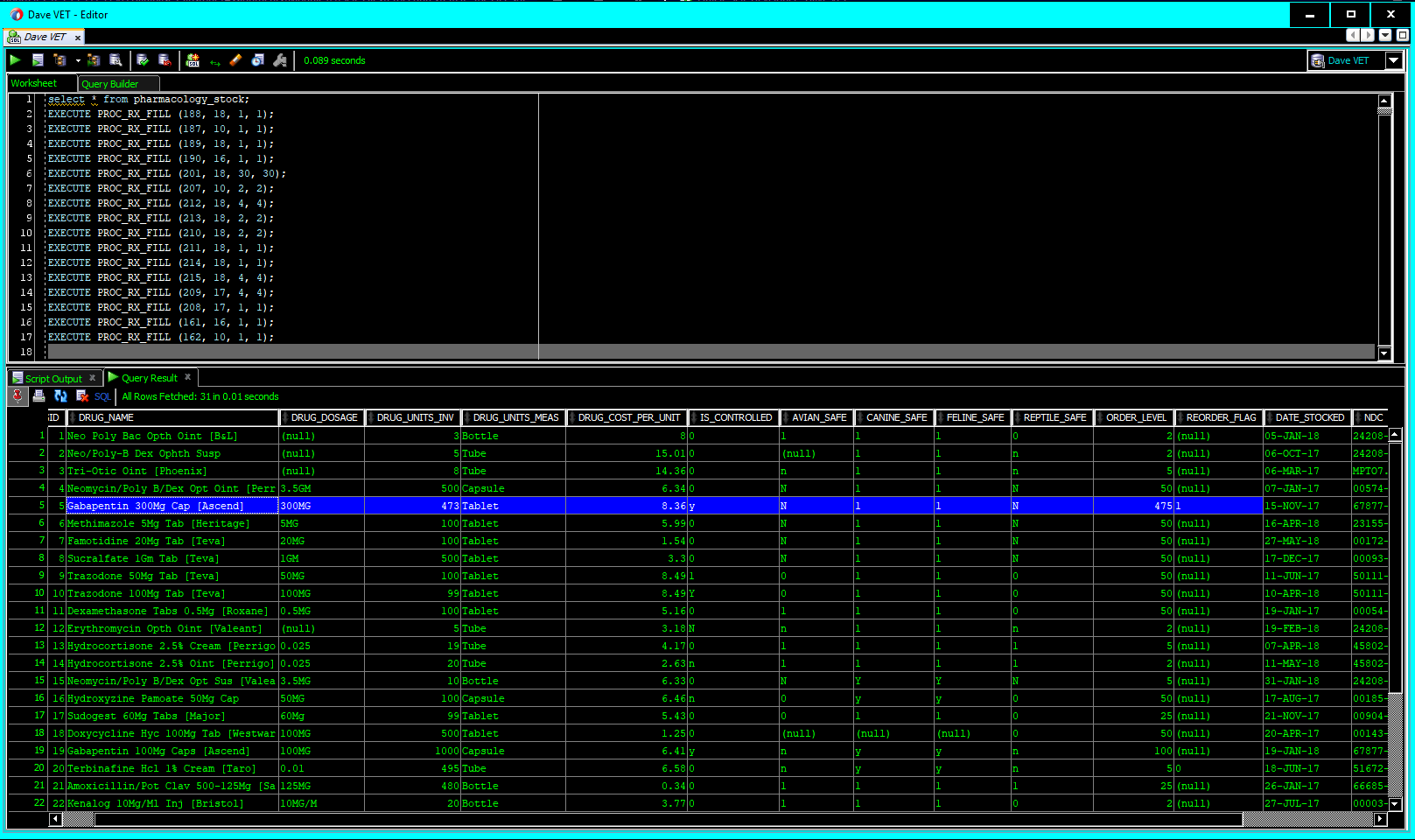
[](https://drive.google.com/open?id=1x3pGkPh2yzmFPvbOlDhXtyorfPGaByOv)

Figure 37

## Patient Chart

The patient chart is the most complex aspect of the business. A simple table listing each prescription, clinical event, etc. for each patient would not only be cumbersome to program, it would lead to significant risk of data integrity and duplication errors. Several tables were created relating to the functions of the clinic and those tables were ultimately combined into meta-views procedurally and with SQL. An overview of the table structure, then an explanation of the procedures, program units and constraints that create the chart follows.

### REPORT-01 Specific information must be included in the charT

The following tables were created to assist with the chart building:

* ANIMAL\_FACTS: this can be thought of as the header for the chart; it contains everything that reception gathers at check-in time such as name, date, an attribute linked to the owner, and the PETID acts as a chart id as no animal would ever have more than one chart at the same clinic. Each one of these child tables is connected to the ANIMAL\_FACTS table through the primary key PETID.
* PATHOLOGY\_HISTORY: this table connects to the pathology lab, and stores results from laboratory tests .
* VET\_PROCEDURE\_HISTORY: in the entity-relationship-diagram included with CSFDAVD, this table was originally called PROCEDURE\_HISTORY: the name was changed to avoid confusion with Oracle’s program units called ‘procedures’. This table stores all the clinical-events, or clinical-procedures that happen with the animal, from a removing a simple bur from a pad, to complex surgeries.
* RX\_HISTORY: has several connections to the other RX and PHARMACOLOGY tables and records all medicines dispensed or prescribed to the patient, including during clinical-procedures.
* RADIOLOGY\_HISTORY: contains images imported from the satellite veterinary radiology business next door, with room for notes about the images.
* IMPORTED\_CHART\_DATA: this contains scanned images, or uploaded .pdf files of patient data from other clinics. Traditionally, this data is kept slightly segregated from data generated by the clinic importing it, (Dr. Hicks, 2018).
* ENCOUNTER\_HISTORY: this table stores items that don’t really go with any other table; for example, the animal’s current weight. This table also contains a CLOB (character large object) field that allows for the vet to make copious notes each time they encounter a pet, that may not be related to a specific lab, medicine, or clinical procedure.

Figure 38 shows these tables highlighted in blue.

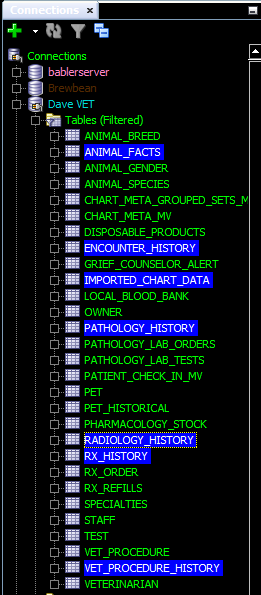
[](https://drive.google.com/open?id=1PFJsMo4d8boj37Gn1yYlkfUWwgVmz4Ii)

Figure 38

The creation of these tables and their relationships satisfied the following requirements:

* CHART-01: Room for notes.
* CHART-02 & CHART-03: All previously and actively used medicines must be shown in the chart.
* CHART-06: Veterinarian must be able to see facts about the animal (weight, gender, species, etc.).
* CHART-07: All medical procedures performed must be stored in the chart.
* CHART-09: Lab work must be stored in the chart.
* CHART-13 & CHART-16: Historical radiology information must be stored in the chart.
* CHART-15: Importing records from other veterinary clinics.
* TRX-12: Save encounter notes to the chart.

### CHART-04 Show all medicines taken by pet easily

A view [RX\_DETAILS\_V] was created to make accessing patient data related to current medication much easier. Functions were used to translate drug, patient, and prescribing veterinarian name into human readable values; if the medication was given during a clinical procedure instead of a prescription, the procedural id was also translated into readable information [figure 39].[](https://drive.google.com/open?id=1Ge_XLn6nPFRL91yL7TmyDWqjXSim_G32)

Figure 39

An additional view was created only showing only maintenance medications for the animal. The view [RX\_HISTORY5YRS\_ALLMAINT\_MEDS\_V] was created to show any medication prescribed and marked as a maintenance medication within the last five years.

### CHART-08 See all known medical procedures for the animal

Like the prescription view, the clinical procedure view [PROCEDURE\_HISTORY\_V] shows English pet, vet, and event names to make searches of the table much more user friendly [figure 40]. This view can also easily be restricted by date with regards to the date of the clinical event, or date where follow up is required (stitches coming out, etc.); thereby also satisfying the requirements for CHART-07.

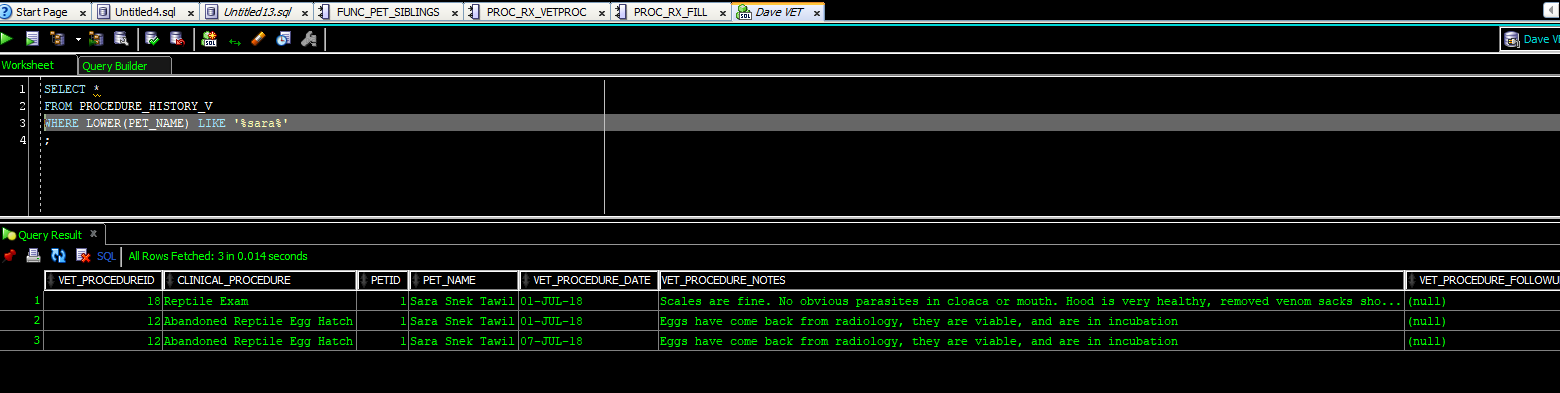
[](https://drive.google.com/open?id=1Yu5pc_QrEAK9P_IS7wp-oBqeclmDKuwB)

Figure 40

### CHART–09 See the most recent lab-work done on the animal

Rather than creating a historical pathology view, and a ‘last-five-tests’ view, a single view [LAB\_WORK\_V] was created that can easily be restricted by date. Much like the prescription and clinical procedure views, the pathology view uses functions to create long form names tied to the keys used in view creation [figure 41].

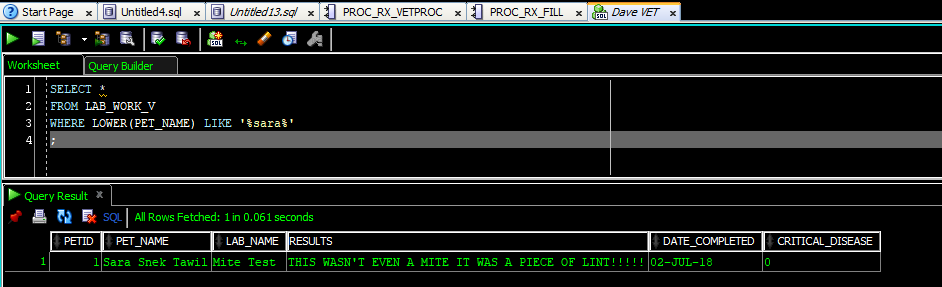
[](https://drive.google.com/open?id=1dz1qoviT2J_fzFgM_hmxqUn76NRa-JtZ)

Figure 41

### CHART-12 Radiology images must be a part of the chart

The last of simple view created was based on the radiology table and was structured like the rest of the chart views. Figure 40 shows this view [RADIOLOGY\_V] along with a saved image showing The patient’s X-Ray. Figure 41 shows a 1.5GB colorized positron emission scan of a cat’s brain. This PET-scan of a cat shows that the tablespace is clearly adaptable enough to handle the massive sizes of radiological imagery files.

[](https://drive.google.com/open?id=1l4VECrT2aAH2xo-Koxo77diFTNvIHXqd)

Figure 42

[](https://drive.google.com/open?id=1CjQQWZnlmEokbZGx0JBJ9ghqAP4DdaE1)

Figure 43

### CHART-15 Imported veterinary data

Rather than entering in the data from other vets into their database Babler’s Veterinary Clinic is taking .pdf scans and storing them in a separate area. The vet will type in relevant notes from the imported data into the chart and the original .pdf scans will be stored on the server. No view was created; however, a table with a BFILE attribute was made to accommodate this need [figure 44]. The app-developers can use this to pull the .pdf up in application from the server if/when the vet needs it.

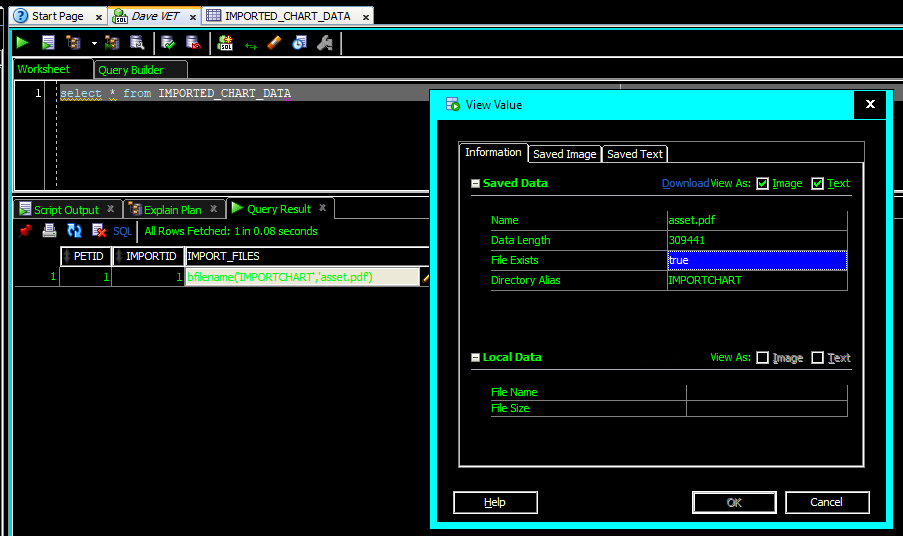
[](https://drive.google.com/open?id=1Zr3UOUnxClZ5jLljl8KvJejszvalrcXv)

Figure 44

### CHART-14 Vet needs to see animal siblings of the patient

During requirements gathering, veterinarians expressed frustration that they had to constantly ask pet parents what other pets were in the house. They need this information on hand immediately in case the patient has a communicable disease, and they need to know if other animals are at risk and need treatment. This information also helps vet determine if an animal is being bullied by other animals or in an environment unsafe for them (you wouldn’t put an uncaged bird with a house full of cats, and you wouldn’t put a snake in a house with a mongoose or a ferret).

Simply listing the species and breed id of the animal’s pet siblings would be providing the data the vet needed, but not the information they are seeking. The function used for patient check-in and grief counselor was modified it so that instead of showing the sibling names, it shows the other types of siblings that live with the patient [FUNC\_ALL\_SIBLING\_BREEDS]. Figure 45 shows the function running versus the pets associated with OWNERID 10. Note how the breed of the pet being called by the function does not show in the field.

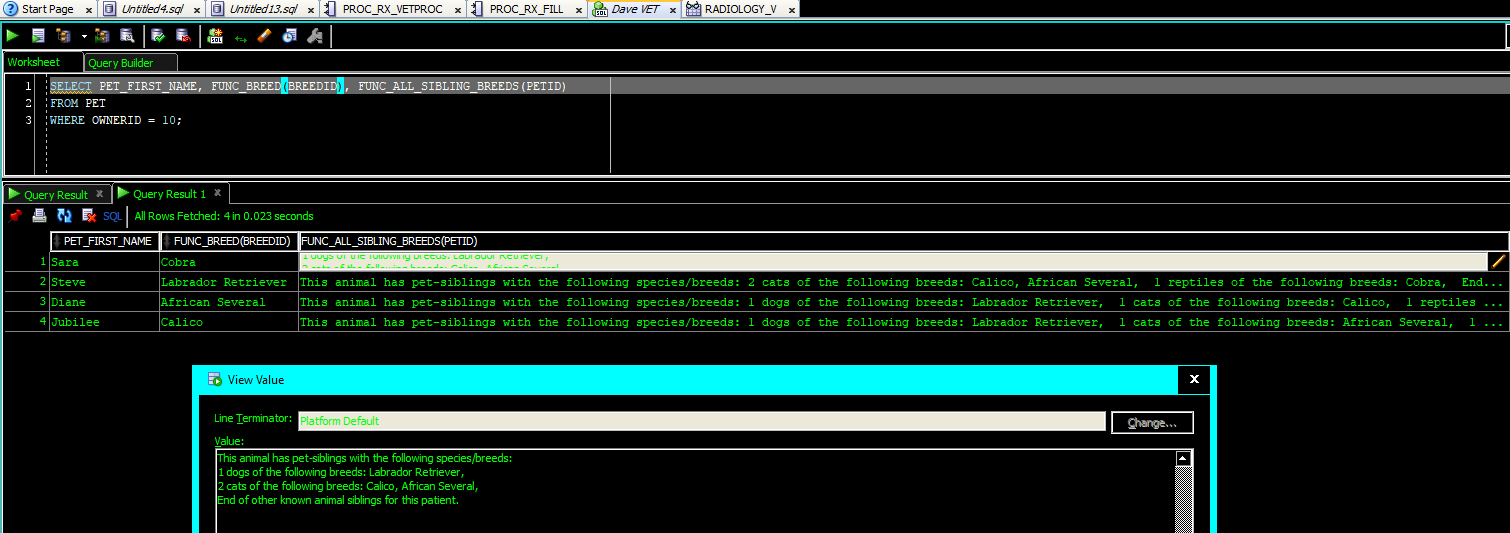
[](https://drive.google.com/open?id=1lv9CYJDSKvj4UkWNzekQxa0uf5NOpTXl)

Figure 45

### CHART-05 Patient notes

Creating a historical readout of patient notes (not related to medication) was a straightforward multi-table UNION. The most significant difficulty was accommodating the CLOB fields. The users felt that 4000 characters would occasionally be insufficient for notes and because of this, all note columns were made into CLOB attributes. This creates a need to code in some conversion functions into the SQL or the view will get rejected. Oracle insists that if one non numeric field is in the union as a CLOB then all must be, even columns that are not touching the CLOB column [code insert 06].

**CREATE** **OR** **REPLACE** **VIEW** CHART\_NOTES\_V

**AS**

**SELECT** PETID **AS** PETID, TO\_CLOB(FUNC\_VET\_NAME(VETID)) **AS** VET, TRUNC(DATE\_COMPLETED) **as** "EVENT\_DATE", TO\_CLOB(FUNC\_LAB\_NAME(LABID)) **AS** EVENT, TO\_CLOB(CRITICAL\_DISEASE) **AS** CRITDISEASE, TO\_CLOB(RESULTS) **AS** NOTES, FUNC\_DUALCLOB('PATHOLOGY') **AS** EVENT\_TYPE

**FROM** PATHOLOGY\_HISTORY

**UNION** **ALL**

**SELECT** PETID, TO\_CLOB(FUNC\_VET\_NAME(VETID)), TRUNC(VET\_PROCEDURE\_DATE) **as** date\_done, TO\_CLOB(FUNC\_PROCNAME(VET\_PROCEDUREID)), TO\_CLOB(**NULL**), TO\_CLOB(VET\_PROCEDURE\_NOTES), FUNC\_DUALCLOB('CLINICAL\_PROCEDURE')

**FROM** VET\_PROCEDURE\_HISTORY

**UNION** **ALL**

**SELECT** PETID, TO\_CLOB(**NULL**), TRUNC(RADIMG\_DATE\_TAKEN), TO\_CLOB(**NULL**), TO\_CLOB(**NULL**), TO\_CLOB(RADIMG\_NOTES), FUNC\_DUALCLOB('RADIOLOGY')

**FROM** RADIOLOGY\_HISTORY

**UNION** **ALL**

**SELECT** PETID, TO\_CLOB(FUNC\_VET\_STAFF(VETID)), TRUNC(ENCOUNTER\_DATE\_TIME), TO\_CLOB(ENCOUNTER\_WEIGHT), **NULL**, TO\_CLOB(ENCOUNTER\_NOTES), FUNC\_DUALCLOB('ENCOUNTER')

**FROM** ENCOUNTER\_HISTORY;

Code Insert 06

Figure 46 shows the view; highlighted in the image are fields that may have been shown in other areas of the chart throughout this document.

[](https://drive.google.com/open?id=1n87uYBVZ_VDGjLiN0M4xythYTWqy4I_e)

Figure 46

## REPORTS REQUIRED

This section contains business rules listed as report requirements in CSFDAVD. These are either additional business rules that need to be presented in plain-text language and other information needs discovered during requirements analysis. Some of the reports from CSFDAVD were satisfied by other requirements and will not be listed here.

### REPORT-04 Medicine safety vs. species check

The previously created view showing prescriptions needing to be filled was found to be insufficient due to the volume of medicines on hand. Pharmacists need to know right away if a medicine is species-safe. Species-safe means the medicine won’t kill the animal; for example, you never give a dog acetaminophen. To ameliorate this issue, another function [FUNC\_RX\_SPECIES\_SAFE] was created that compares attributes in the PHARMACOLOGY\_STOCK table to the SPECIESID of the animal, then the function was added to the view [figure 47].

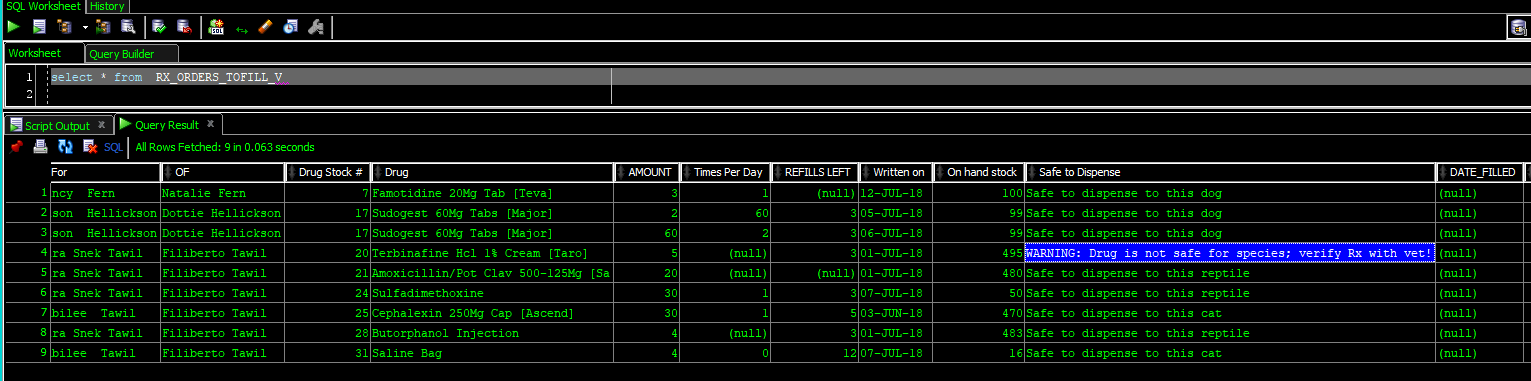
[](https://drive.google.com/open?id=15-ESgXvhreuTGX6XyOgrRJhgOKzECS72)

Figure 47

### REPORT-02 Pharmacists need to see inventory information about drugs

The chemists and pharmacists want to be able to see a daily report that shows them what is on hand at the beginning of the day and what is close to expiring. A materialized view was created for them that refreshes daily [figure 48].

[](https://drive.google.com/open?id=1s5g9jJguodeKfLklIXf-t2Q1VBjnvyrG)

Figure 48

### REPORT-01 Natural language chart

Traditionally a natural language document (think Microsoft Word) would be handled at the application level, instead of in the RDBMS; however, to demonstrate the abilities of PL/SQL and to provide the application programmers options, exemplary natural language charts were created in the database.

First a chart header was created as a view [CHART\_HEAD\_V], this data contains the basic information that a vet would require immediately when walking into an examination room [figure 49].[](https://drive.google.com/open?id=1qEkmbzIY3m_RGvw3xI_o0Q5YG-n5nnpt)

Figure 49

Then a function was created to make the header more natural for a user to read. The function is generated by the PETID [figure50].

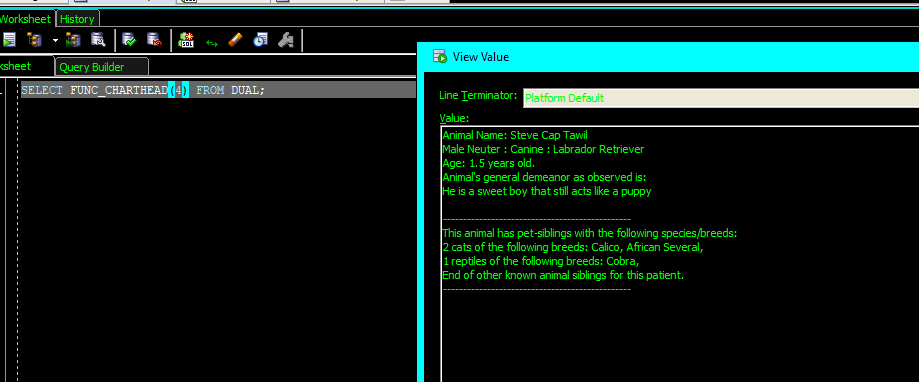
[](https://drive.google.com/open?id=1ibz1pnGixbAJsIorg0SENOHL9p9gsBbJ)

Figure 50

Similar functions were created for chart notes, and prescription information [figures 51 & 52].

[](https://drive.google.com/open?id=1MMX-nrWCYZVvIELtIRQVIRkFdTRyCLm5)

Figure 51

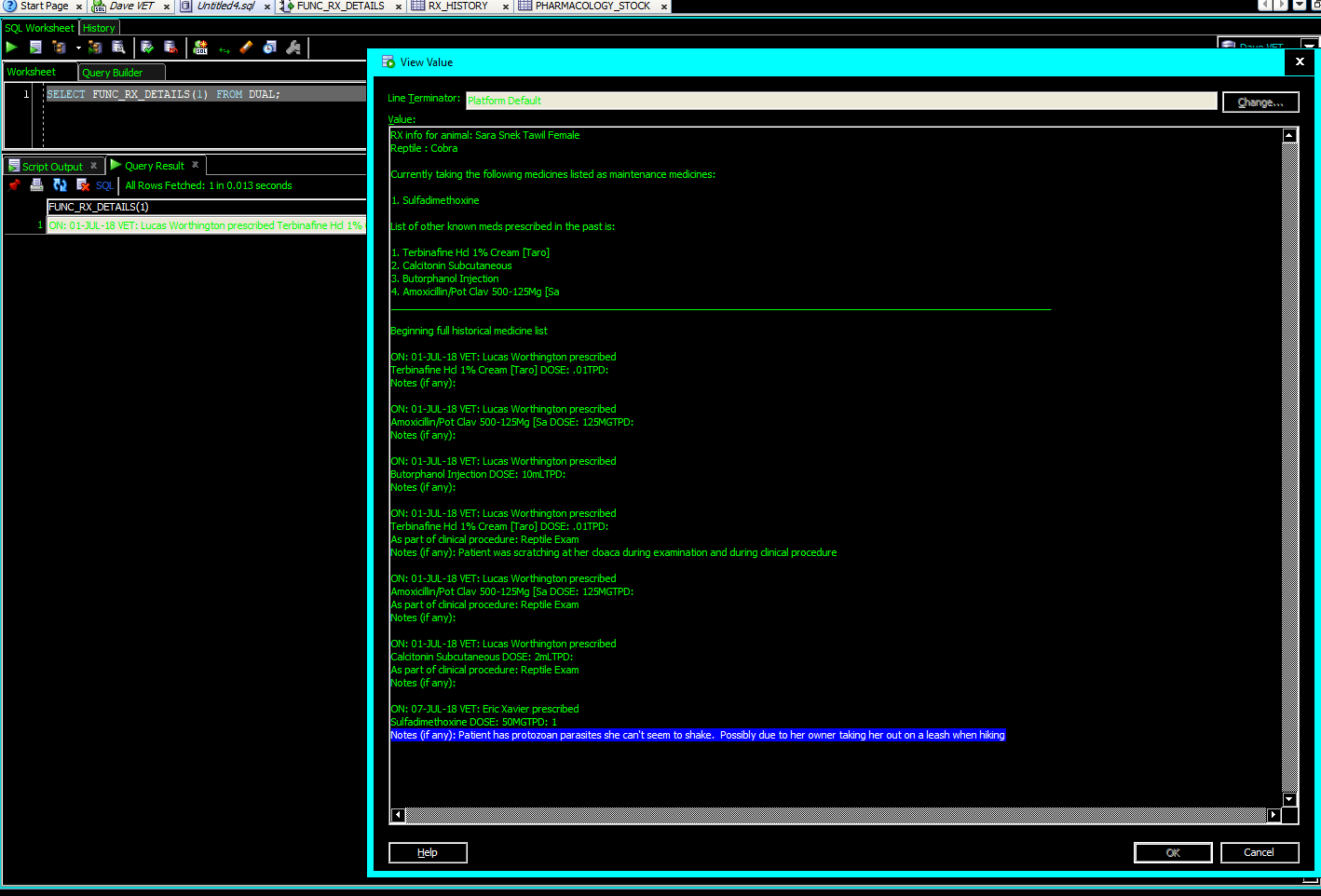
[](https://drive.google.com/file/d/1-K-Rqak91GaH8LuHYJwp5VTJn_i1JF5N/view?usp=sharing)

Figure 52

These functions were then combined into a view that creates a single field connected by PETID allowing for a full chart readout [figure 53].

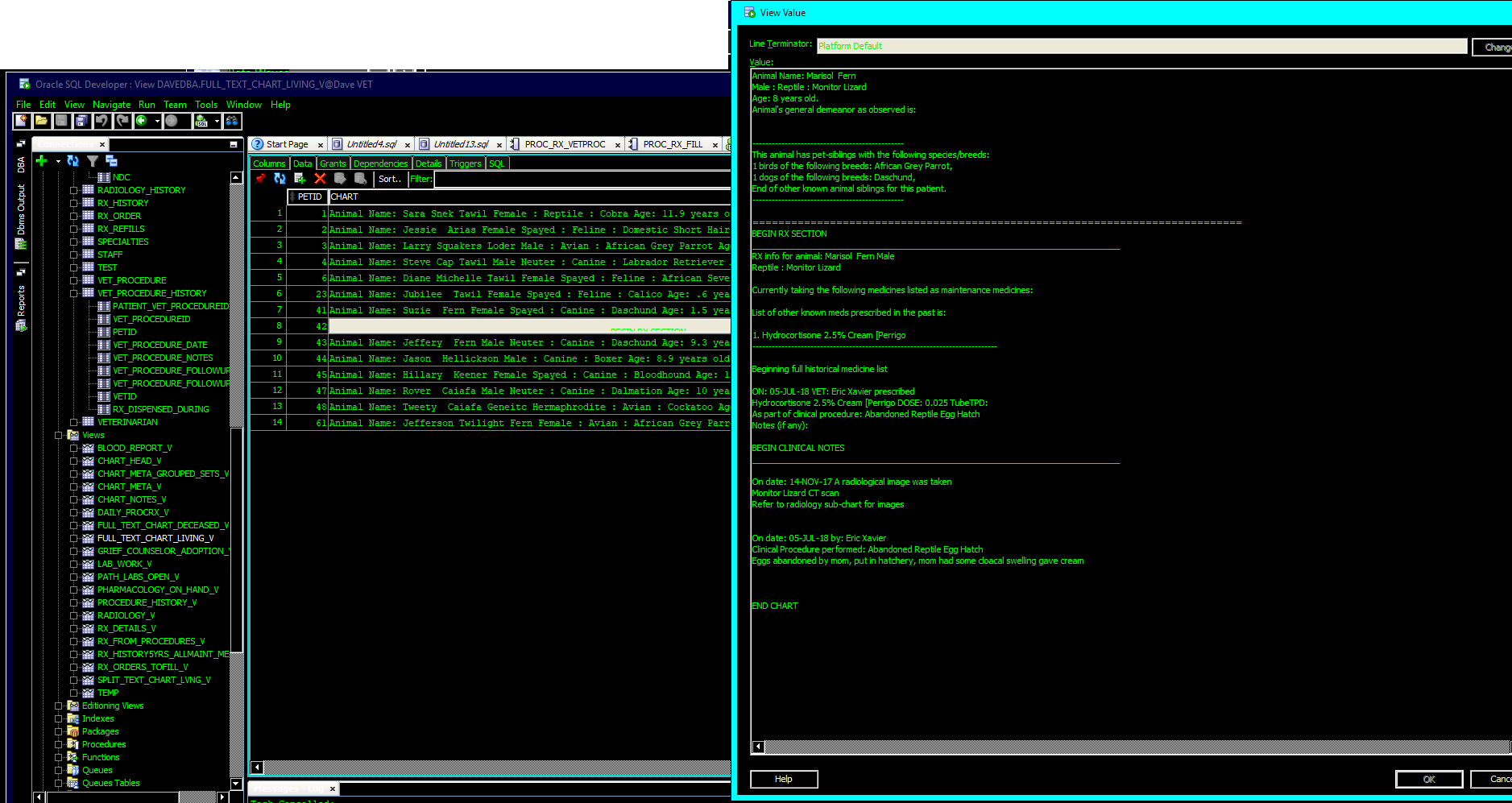
[](https://drive.google.com/open?id=1YpYCKdNYgHfGfY9cIkkyn0rBAOHlANYr)

Figure 53

All these functions and procedures were wrapped in an Oracle Package [CHART\_PKG].

#### REPORT-01: PART 2 Traditionally joined chart

I also created a traditional view involving a series of joins on the chart. I started with a Materialized View that updates once a day, and then used plain English functions to translate the keys [figure 54][[8]](#footnote-8).

[](https://drive.google.com/open?id=1-i6Js-9tNHMBRF3FS4RjiKl5B-vythmm)

Figure 54

The table is set up so that the important information in the chart header is still included with each row, and so that each individual prescription, procedure and, pathology lab shows on the chart. Because there will only ever be one encounter per day (anything else would be an examination, and thus a clinical procedure) encounters are thus restricted to one occurrence in the where clause for [CHART\_META\_MV] which feeds [CHART\_META\_V].

### REPORT-ADDENDUM Searching capabilities

The example in [Figure 54] is more readable, but not quite as useful for searching. Especially since everything is stored as a CLOB.

To assist programming with performing searches a procedure was created, and a functionality of Oracle was explained to programming.

#### CLOB search functionality in oracle.

Searching through CLOB files is simple with SQL in Oracle; though some are intimidated by Oracle’s dialect of SQL. To assist the programming team, an example was provided [Figure 55].

[](https://drive.google.com/open?id=1aH59vBaWFRivsNEpP43QF2Wlzn0izHhK)

Figure 55

#### Search procedure for the traditionally JOINed chart

A search procedure was also created for searching the traditionally JOINED chart and allows for using the condition ‘LIKE’ which creates a pseudo-fuzzy search capability. The procedure [PROC\_CHART\_SEARCH] was created to accept between one and three terms. Figure 56 shows the procedure using one term and figure 57 shows the procedure using three terms.

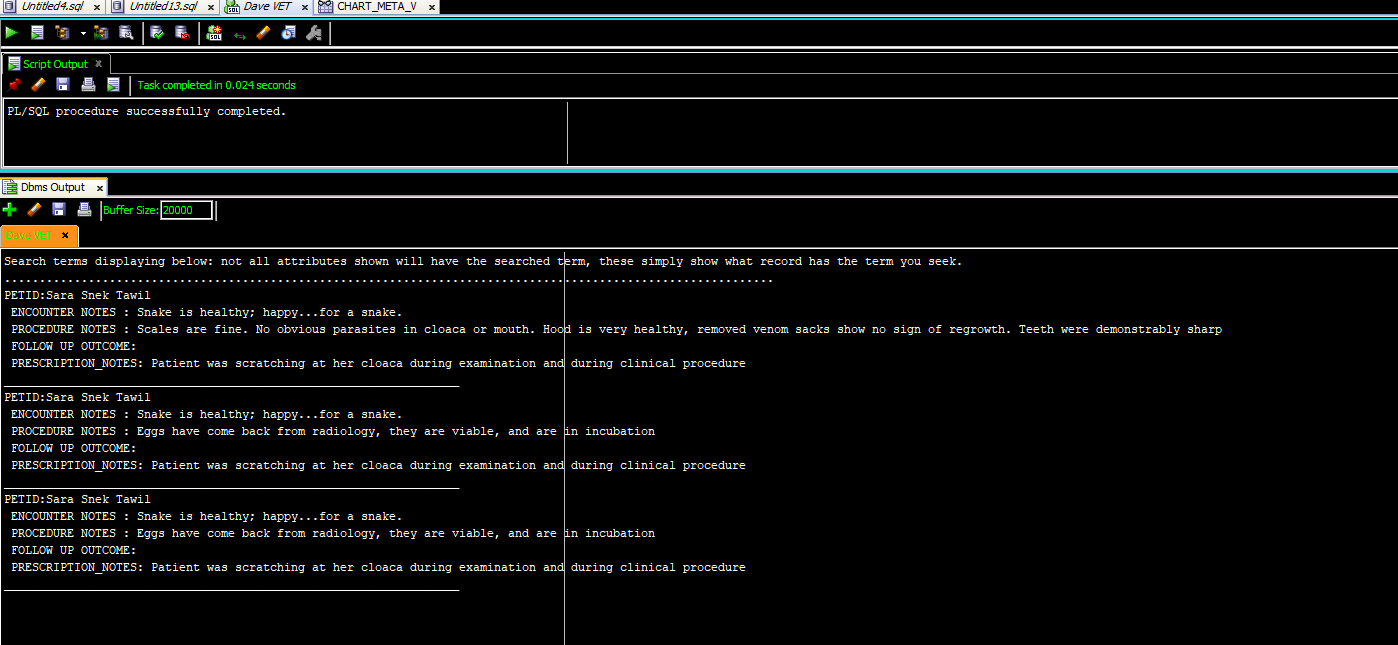
[](https://drive.google.com/open?id=1nhKTvTqCNs8jD2O2Gi95bK_yQqNRPBwK)

Figure 56

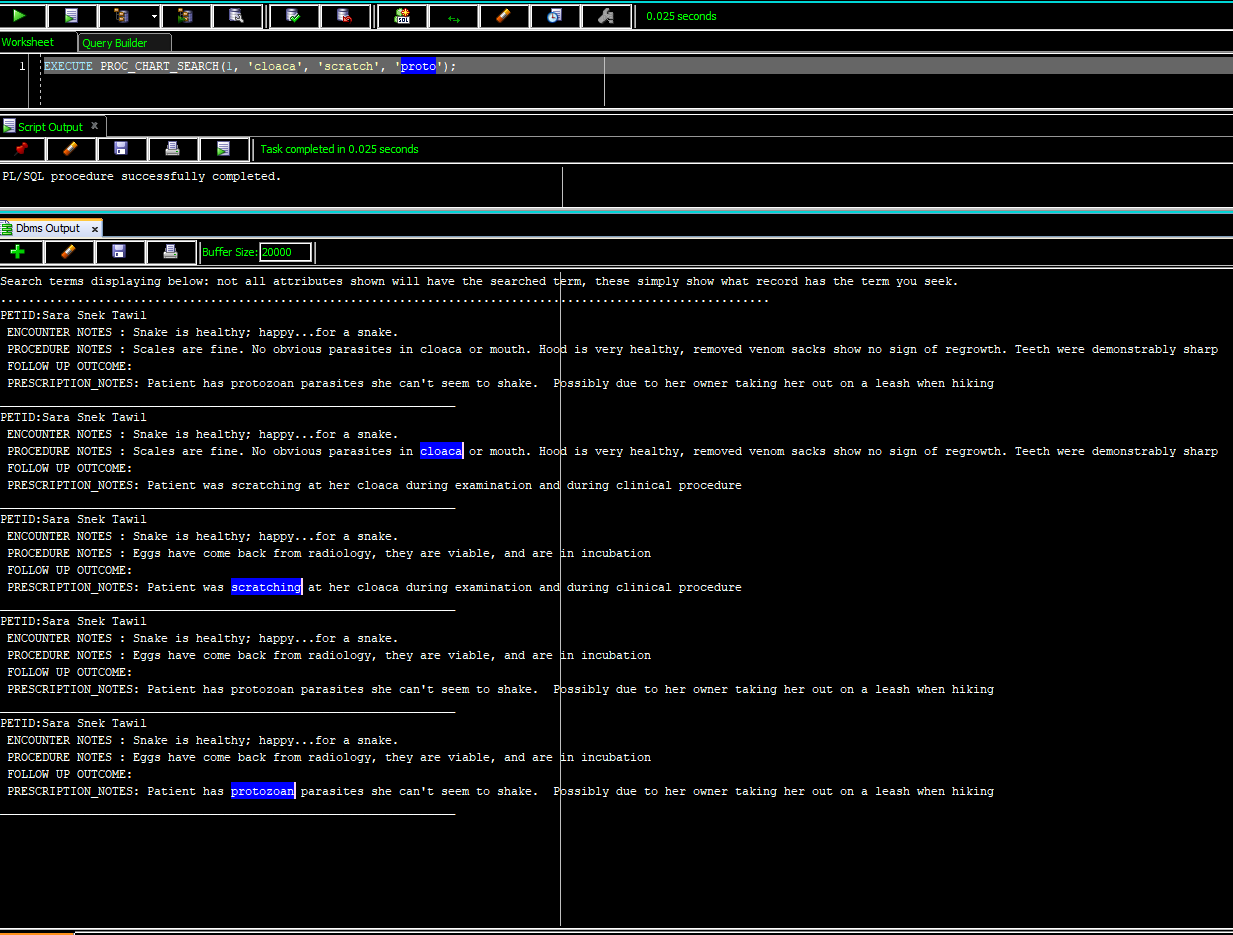
[](https://drive.google.com/open?id=1MWxFKnpUaFmsryG5upIrjqfuQq5ZBLnJ)

Figure 57

# MISCELLANEOUS TRANSACTIONS, REPORTS, AND VIEWS

## TRX-03 Inventory Disposable equipment

Client wished to have a table to inventory disposable equipment; a table was created to satisfy this need [DISPOSABLE\_PRODUCTS].

## Blood Bank

### TRX-01 Keep track of blood inventory

A table was created to satisfy this need [LOCAL\_BLOOD\_BANK].

### TRX-11 See AVAILABLE blood

Customer wanted to see how much blood they have on hand, and how much per species. To satisfy this need I was able to create a view using Oracle’s powerful GROUP BY CUBE function as shown in figure 55. A NVL function was created to fill in the top row as “Grand Total” instead of (null) to avoid confusion.

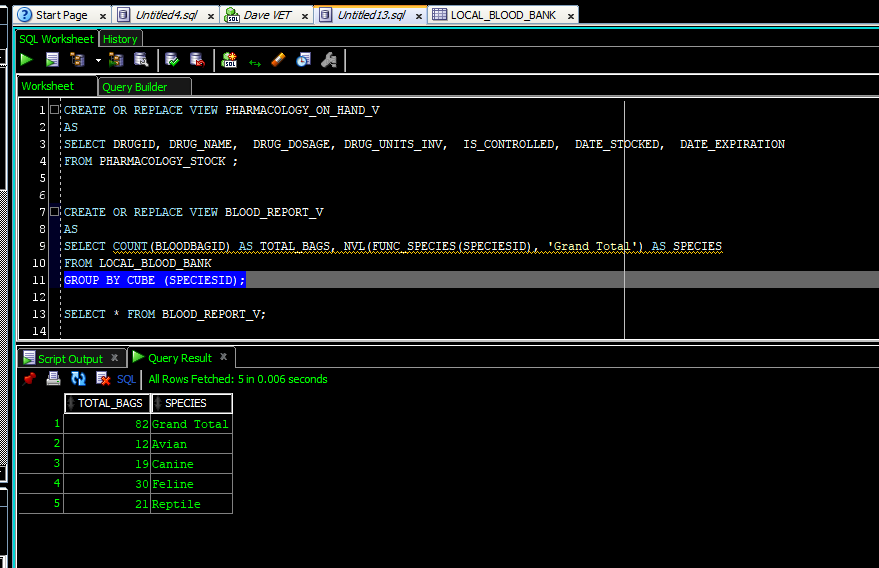
[](https://drive.google.com/open?id=1bJPWORK6yFHXqii6d15vbtWAuOE8WPGw)

Figure 58

## Not included rules

As with any project, requirements change or become obsolete during the building phase. The following anticipated business rules, transactions or reports were not included:

* TRX-14: Patient Chart Global View—became obsolete after natural text and traditional tables were created.
* REPORT-04: Pharmacology by type—numerous efforts were made to get the pharmacists to explain what they meant by type. It was unclear if they meant type as in: capsule, liquid, tablet, injection etc.; or did they mean type like: antibiotic, chemotherapeutics, antihistamines, NSAID, opioids, nerve-agents, etc. The chemists and pharmacists simply did not make the time to specify this need. After numerous attempts to obtain this information it was decided this need will not be included in this iteration of the database.

# PERFORMANCE TUNING

The use of highly normalized tables, (i.e., tables with minimized data redundancies) reduces some of the needs for performance tuning as they already implement a fairly high ratio of keys to attributes(Coronel & Morris, 2017). Key indexes are quite helpful, but alone are ultimately not sufficient for a live database. Indexes were added in based on discussions during requirements gathering regarding what would be the most queried areas of the database [figure 56]. The indexes on the tables that make up the chart are dual attribute indexes of PETID and that table’s date attribute (for example PETID and DATE\_PRESCRIBED). This type of indexing was chosen in anticipation of vets needing to search or sort by a patient’s events within a specific date range.

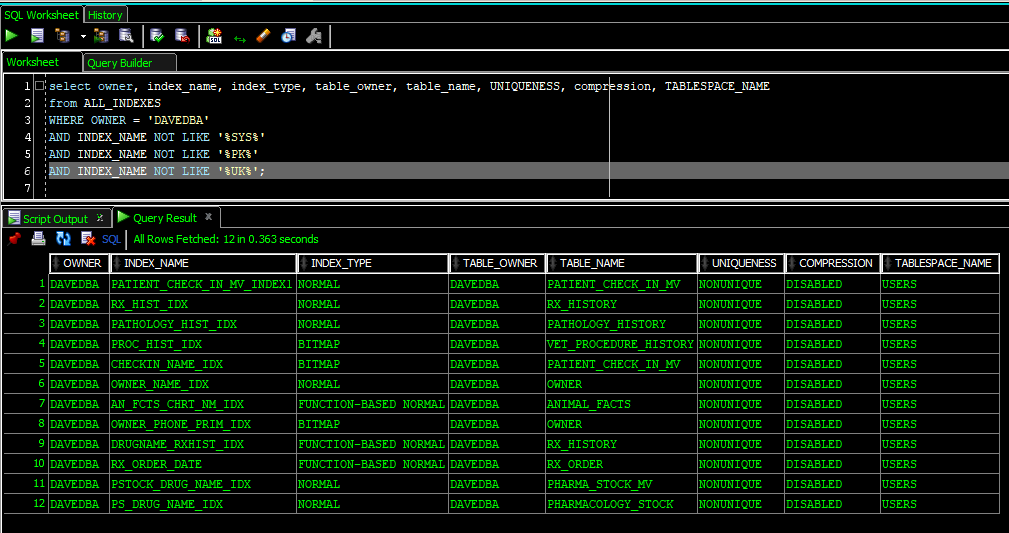
[](https://drive.google.com/open?id=1oi8eBx7CeV7NuVO0bhZwjmTDytgVb4Mj)

Figure 59

# DATA STRUCTURES AND PROGRAM UNITS

This section shows the list of data structures and program units in the database; through SQL Developer’s directory-style navigation, and then through data dictionary views.

## Tables, Views, Materialized Views

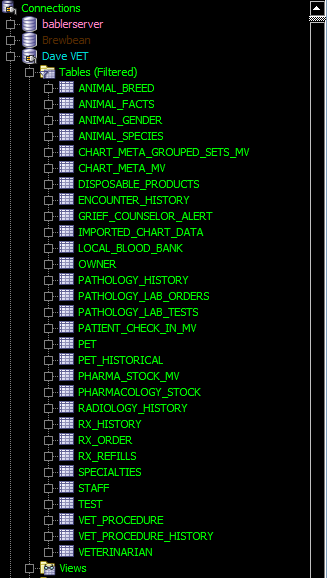


Figure 60

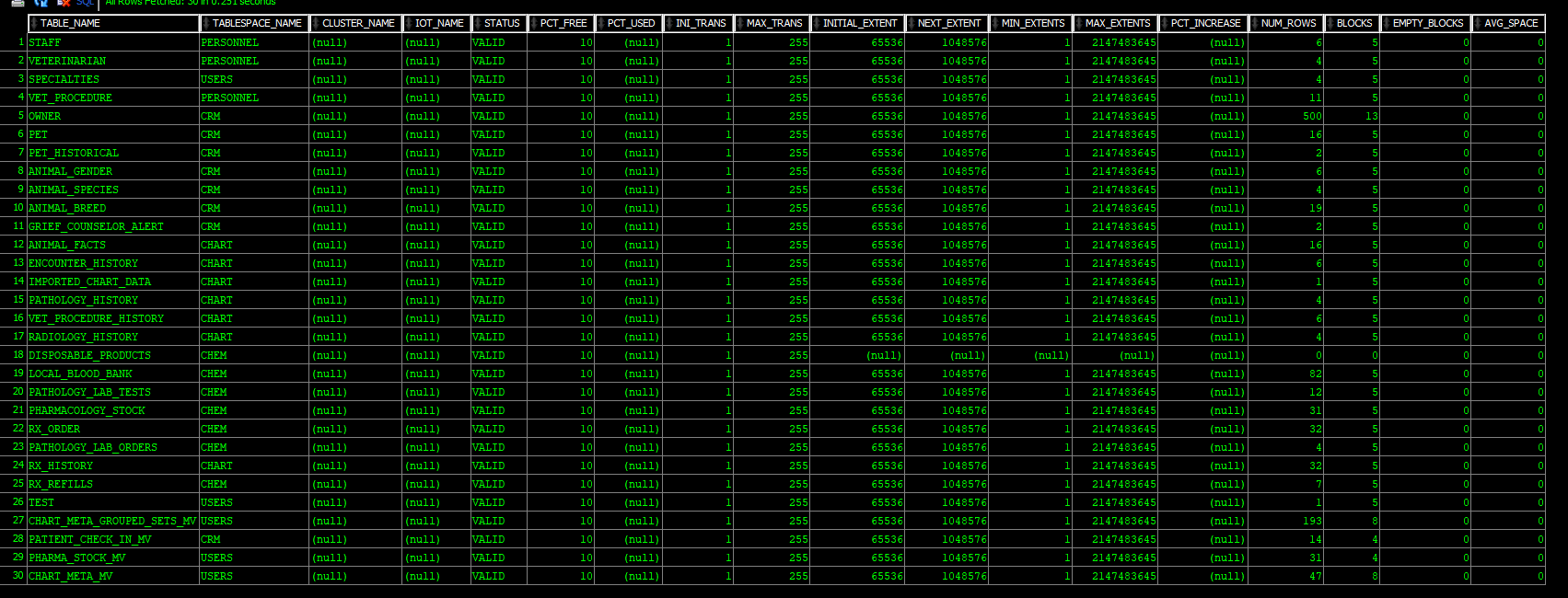
[](https://drive.google.com/open?id=1hWO5RwclI3slshx-MyP6JVYqmhL1SSPX)

Figure 61

# BIBLIOGRAPHY

|  |
| --- |
| ASSM - Oracle FAQ. (n.d.). Retrieved October 15, 2016, from http://www.orafaq.com/wiki/ASSM |
|  |
| ASSM, freelists and PCTFREE. (n.d.). Retrieved October 15, 2016, from http://www.dba-oracle.com/t\_assm\_freelists\_pctfree.htm |
|  |
| Babler, Dave Case Study for Developing a Veterinary Database (2018, June 12). Unpublished. Department of Computer Science, Hillsborough Community College, Tampa, FL. |
|  |
| Burleson, D. (2003, January 13). Oracle9i’s automatic segment space management improves segment storage internals. Retrieved October 16, 2016, from http://www.techrepublic.com/article/oracle9is-automatic-segment-space-management-improves-segment-storage-internals/ |
|  |
| Data Blocks, Extents, and Segments. (n.d.). Retrieved October 15, 2016, from https://docs.oracle.com/cd/A57673\_01/DOC/server/doc/SCN73/ch3.htm |
|  |
| Database Concepts. (n.d.). Retrieved September 29, 2016, from https://docs.oracle.com/cd/B19306\_01/server.102/b14220/physical.htm |
|  |
| Dr. J.K. Hicks. (2018, May 31). Interview with clinical pharmacologist covering in general how some human related medical reporting may relate to veterinary needs. [In Person]. |
|  |
| Hall, T. (n.d.). ORACLE-BASE - Oracle Automatic Segment Space Management. Retrieved October 15, 2016, from https://oracle-base.com/articles/9i/automatic-segment-free-space-management |
|  |
| Hass, M. (2015, February 10). Part 1 of 3: Oracle Database Compression. Retrieved October 15, 2016, from http://www.mythics.com/about/blog/part-1-of-3-oracle-database-compression |
|  |
| Jaquier, Y. (2010, May 1). Data compression with Oracle 11gR2. Retrieved October 15, 2016, from http://blog.yannickjaquier.com/oracle/data-compression-with-oracle-11gr2.html |
|  |
| Managing Tablespaces. (n.d.). Retrieved October 15, 2016, from http://docs.oracle.com/cd/B10501\_01/server.920/a96521/tspaces.htm#19171 |
|  |
| Mustafa, O. (2012, October 22). Osama Mustafa Oracle Blog: AWR Reports Vs ASH Reports. Retrieved February 5, 2017, from http://osamamustafa.blogspot.com/2012/10/awr-reports-vs-ash-reports.html |
|  |
| Oracle. (2012, January). An Oracle White Paper Advanced Compression with Oracle Database 11g. Retrieved October 15, 2016, from http://www.oracle.com/technetwork/database/focus-areas/storage/advanced-compression-whitepaper-130502.pdf |
|  |
| Oracle DBA Forums > Maunal & Automatic Segment Space Management (ASSM). (n.d.). Retrieved October 15, 2016, from http://dbaforums.org/oracle/lofiversion/index.php?t21461.html |
|  |
| Reclaiming Wasted Space. (n.d.). Retrieved October 16, 2016, from https://docs.oracle.com/cd/E18283\_01/server.112/e17120/schema003.htm |
|  |
| Rittman, M. (2008, September 14). Testing Advanced (OLTP) Compression in Oracle 11g. Retrieved October 15, 2016, from http://www.rittmanmead.com/blog/2008/09/testing-advanced-oltp-compression-in-oracle-11g/ |
|  |
| Ruel, C., & Wessler, M. (2014). Oracle 12c for dummies. Hoboken, NJ: John Wiley & Sons. Retrieved from http://www.123library.org/book\_details/?id=114851 |
|  |
| Singh, D. (2012, April 11). DHARAMDBA: EXTENT MANAGEMENT and SEGMENT SPACE MANAGEMENT. Retrieved October 15, 2016, from http://dharamdba.blogspot.com/2012/04/extent-management-and-segment-space.html |
|  |
| sql - Removing duplicates from JOIN between two tables (Oracle) - Stack Overflow. (n.d.). Retrieved July 14, 2018, from https://stackoverflow.com/questions/8590552/removing-duplicates-from-join-between-two-tables-oracle |
|  |
| SQL Server vs. Oracle: Output table or multiple rows from procedure. (n.d.). Retrieved July 14, 2018, from http://sqlbisam.blogspot.com/2013/12/output-table-or-multiple-rows-from-stored-procedure-function.html |
|  |
| Tablespaces and Datafiles. (n.d.). Retrieved September 29, 2016, from https://www.csee.umbc.edu/portal/help/oracle8/server.815/a67781/c03space.htm |
|  |

*All images not taken by the author are covered under fair-use doctrine.*

# ACKNOWLEDGEMENTS

I would first like to thank Professor Larry Bross for his excellent guidance and mentorship through my database classes. I would also like to thank Dr. Cameron Spears for the structured and object programming instruction I received from him in his classes which were invaluable to have as a base when learning PL/SQL from Professor Bross. I’d also like to thank my spouse supporting me through my educational endeavors. Finally, I would like to thank all my companion animals past, present, and future. My pets were a great inspiration to me for this project. Without my pets I never would have considered this type of business for my capstone; and, I never would have had the opportunity to observe the functions of a veterinary practice with, at times, alarming frequency.

# APPENDIX A: SQL FILES

#### TABLES & DATA STRUCTURES

1. Full text definitions and a table of all business rules, transactions and report requirements can be found in CSFDAVD. [↑](#footnote-ref-1)
2. Business rules will be grouped however, they may not necessarily be presented in numerical order. [↑](#footnote-ref-2)
3. I strongly recommend right clicking and then choosing open link to see images in their full size in a browser. [↑](#footnote-ref-3)
4. Unless noteworthy no further code will be directly embedded into this document; however, all code will be attached as an appendix. [↑](#footnote-ref-4)
5. At the time of procedure creation we had not yet convinced the lead vet that PETID was effectively CHARTID this was handled at a later time and B.R. CHART-10 was nullified. [↑](#footnote-ref-5)
6. Reminder: all images are referenced to an online version and can be viewed in a browser by clicking <Ctrl> + Click in Microsoft Word. [↑](#footnote-ref-6)
7. Oracle’s PL/SQL does have a Boolean data type, but that is not helpful during table creation. [↑](#footnote-ref-7)
8. Reminder: <Ctrl> + Click in word will pop this out in a browser window [↑](#footnote-ref-8)