Dave Babler

Supervised by: Professor Larry Bross

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

# TABLE OF CONTENTS

[TABLE OF CONTENTS 2](#_Toc521249293)

[DOCUMENT PURPOSE 3](#_Toc521249294)

[WARNING 3](#_Toc521249295)

[NOTE 3](#_Toc521249296)

[INSTALLATION 3](#_Toc521249297)

[DATA STRUCTURE CREATION 5](#_Toc521249298)

[Tablespaces 5](#_Toc521249299)

[Tables and Constraints 7](#_Toc521249300)

[Sequences and Identity Columns 10](#_Toc521249301)

[Data Loading 11](#_Toc521249302)

[BUSINESS RULES IMPLEMENTATION 11](#_Toc521249303)

[Record Keeping, Reception, Customer Relations Managment 11](#_Toc521249304)

[Clinical Procedures 18](#_Toc521249305)

[Pharmacology, Pathology, Blood Bank 23](#_Toc521249306)

[RX-10 a separate id for each individual prescription external to the chart 26](#_Toc521249307)

[RX-02 Stock reorder flag 26](#_Toc521249308)

[Patient Chart 28](#_Toc521249309)

[REPORTS REQUIRED 34](#_Toc521249310)

[INVOICING & ESTIMATE CREATION 42](#_Toc521249311)

[MISCELLANEOUS TRANSACTIONS, REPORTS, AND VIEWS 42](#_Toc521249312)

[TRX-03 Inventory Disposable equipment 42](#_Toc521249313)

[Blood Bank 42](#_Toc521249314)

[Not included rules 43](#_Toc521249315)

[PERFORMANCE TUNING 43](#_Toc521249316)

[DEPENDENCY ANALYSIS 44](#_Toc521249317)

[DATA STRUCTURES AND PROGRAM UNITS 46](#_Toc521249318)

[Tables, Views, Materialized Views 46](#_Toc521249319)

[Program Units 49](#_Toc521249320)

[Sequences & Indices 53](#_Toc521249321)

[BIBLIOGRAPHY 56](#_Toc521249322)

[ACKNOWLEDGEMENTS 58](#_Toc521249323)

[APPENDIX A: SQL FILES 59](#_Toc521249324)

# 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 [Figure 1-Figure 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.

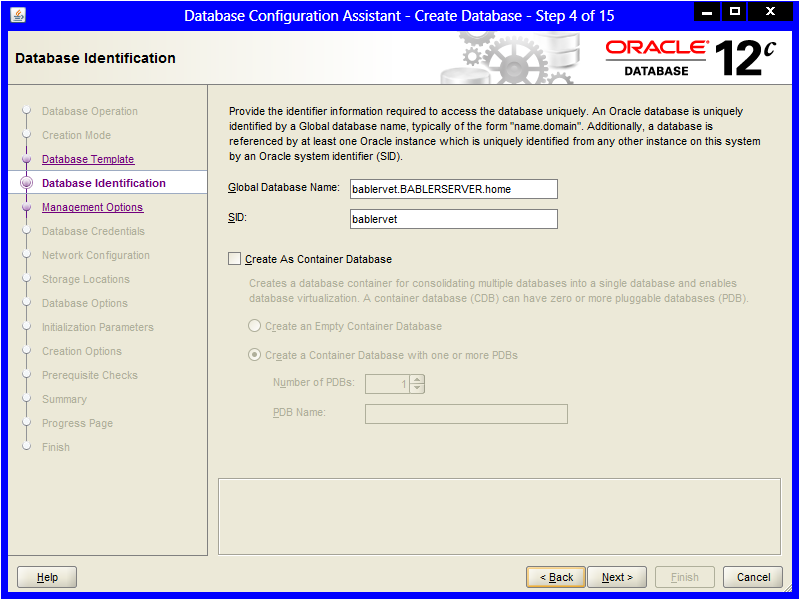
[](https://drive.google.com/open?id=1wMlQ5XdY5jkWAkY-da4BKDEOPqJYjtl1)

Figure 1

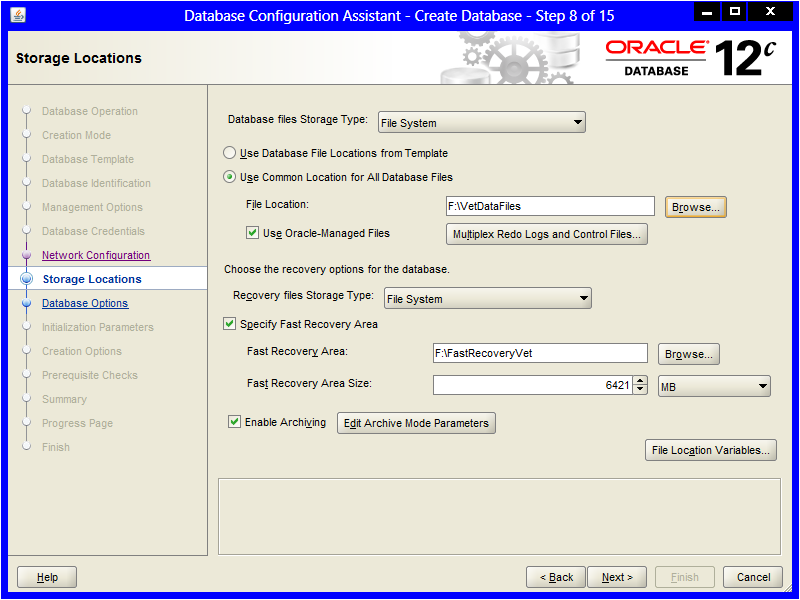
[](https://drive.google.com/open?id=1df7WocTjV8p7xbNkTMlj9Zfl_feYpXQ0)

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 [Figure 4 &Figure 5].

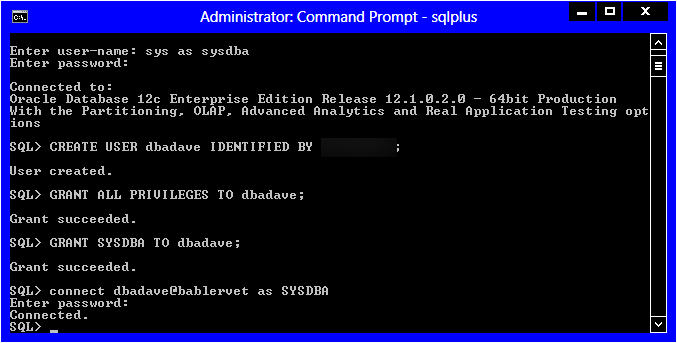
[](https://drive.google.com/open?id=1vv9n3M84h9qOu7zVNgWyb_XOPU5QeFzj)

Figure 4

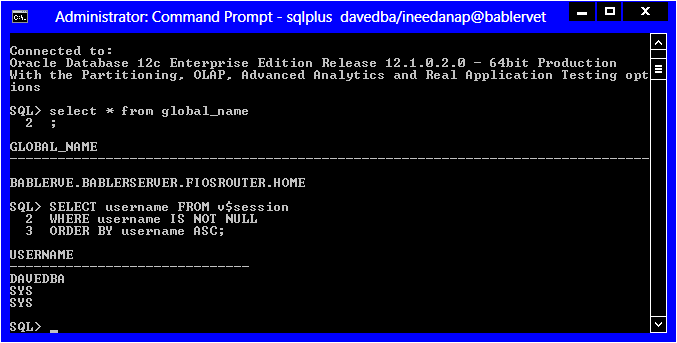
[](https://drive.google.com/open?id=11e4gRLXgM88yuYxgXAKBLh6bFLApEEc2)

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 Figure 6 and Figure 7.

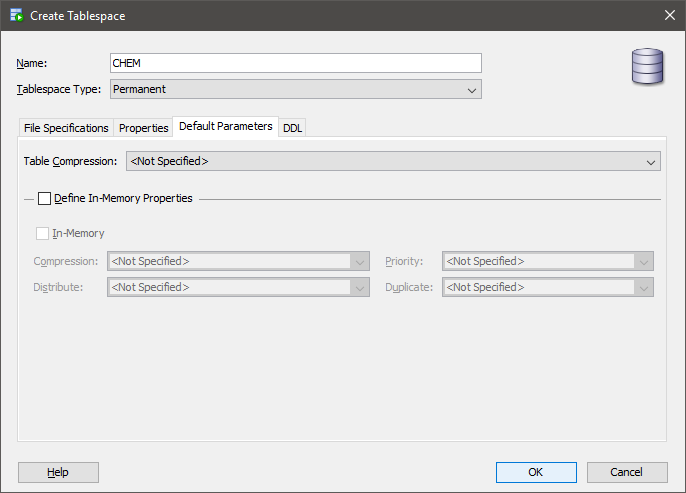
[](https://drive.google.com/open?id=1CJ_4Tyen9oldoZW3y360CyHyEWYMcTsD)

Figure 6

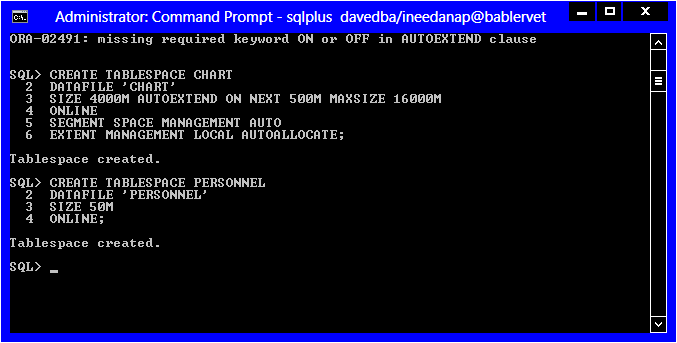
[](https://drive.google.com/open?id=1vJbqhdLpK9X-l8HNjpnPqWWZV5bAwOZ2)

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.

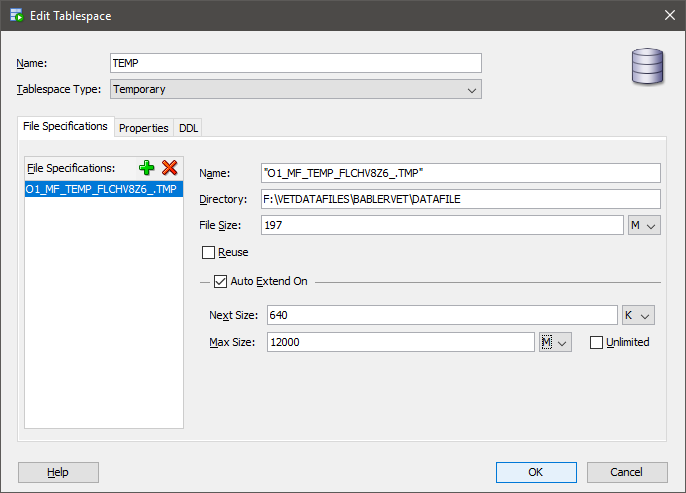
[](https://drive.google.com/open?id=17yn2MR6hrkt58wOeu7PClWbvKljZM02w)

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].

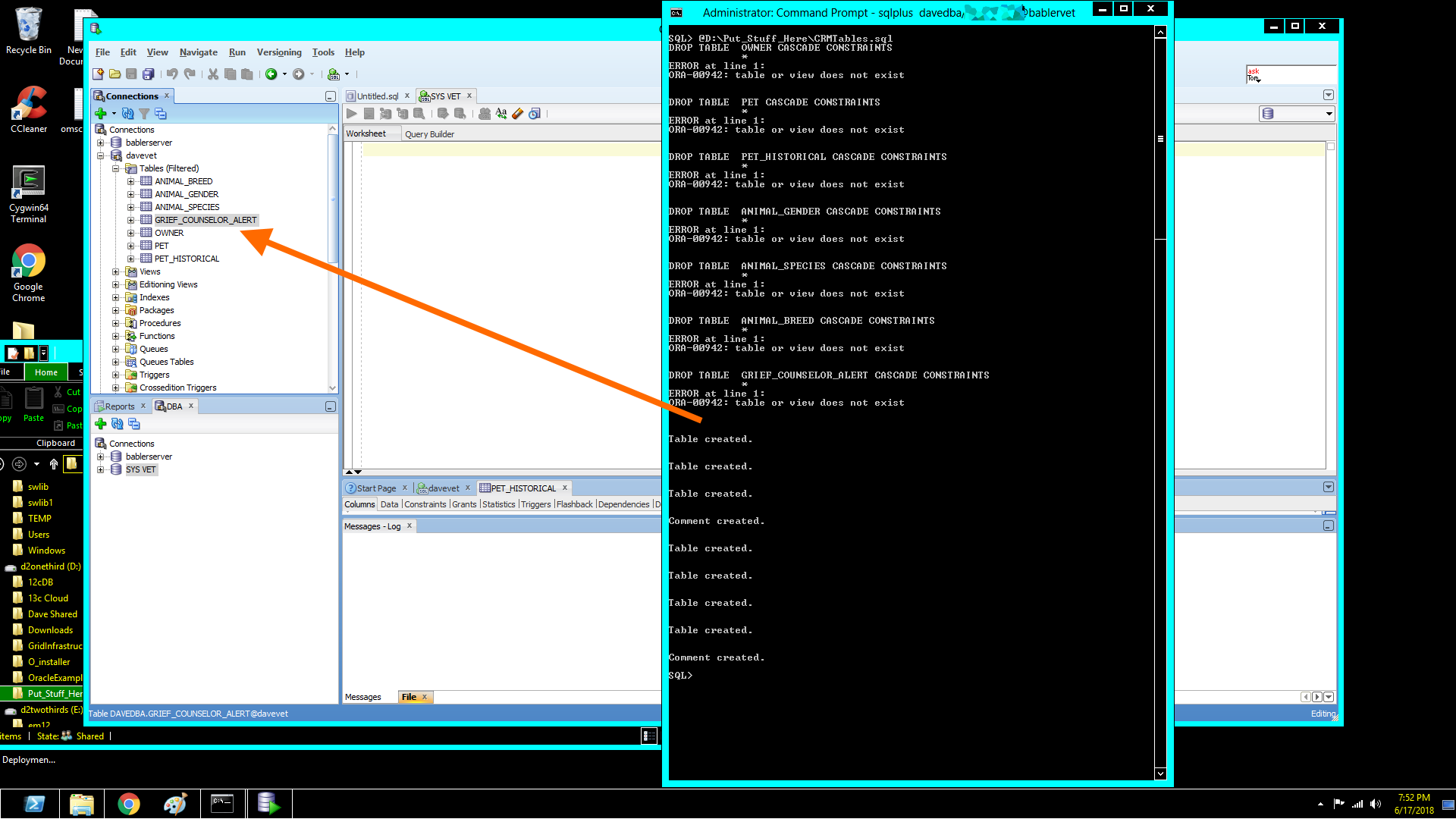
[](https://drive.google.com/open?id=1sPEWK_IDQ-Da93nKyHvWAB2Sc5xqvwjU)

Figure 9

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

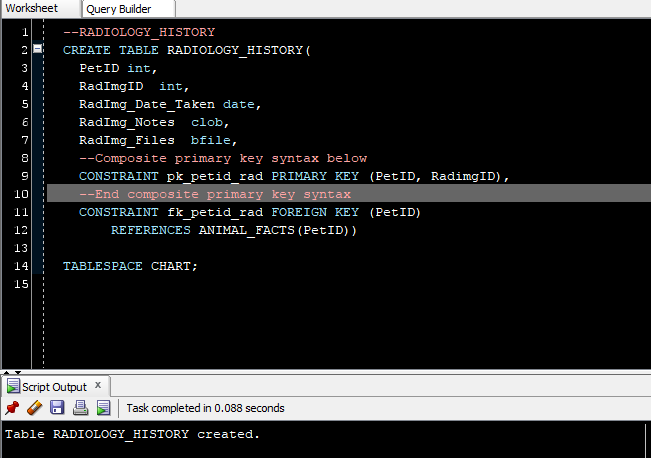
[](https://drive.google.com/open?id=1AEYkpo0_3yqFe4xFdT8scMdnxN_OeCao)

Figure 10

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

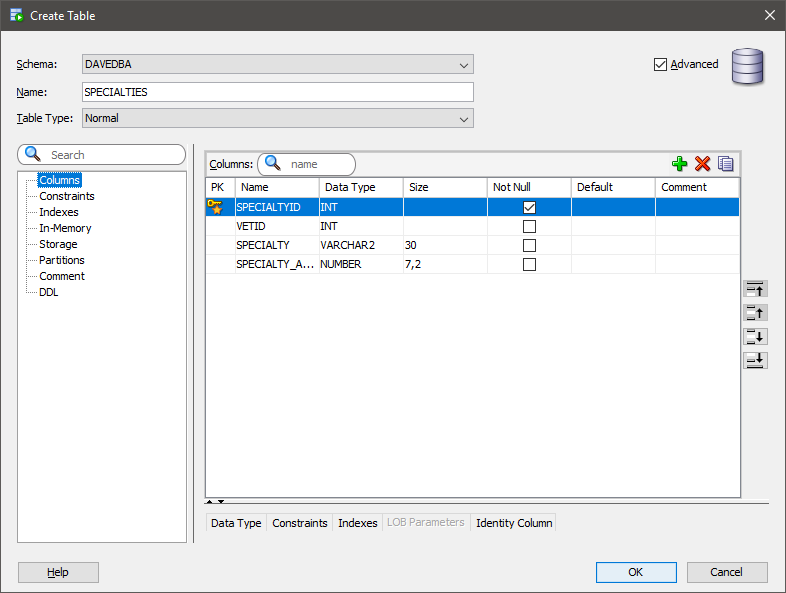
[](https://drive.google.com/open?id=1u8P7EEIXItnZzRSUYx9fZozEwPFWCIja)

Figure 11

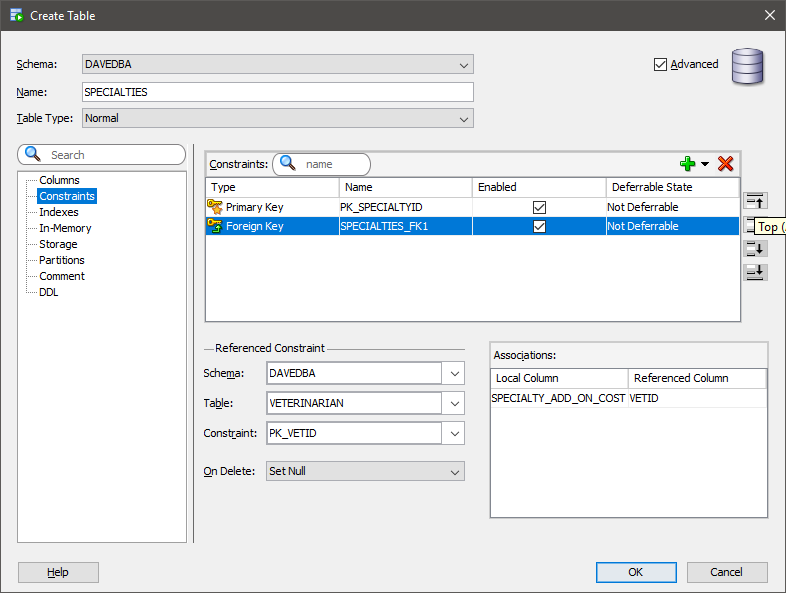
[](https://drive.google.com/open?id=1u8P7EEIXItnZzRSUYx9fZozEwPFWCIja)

Figure 12

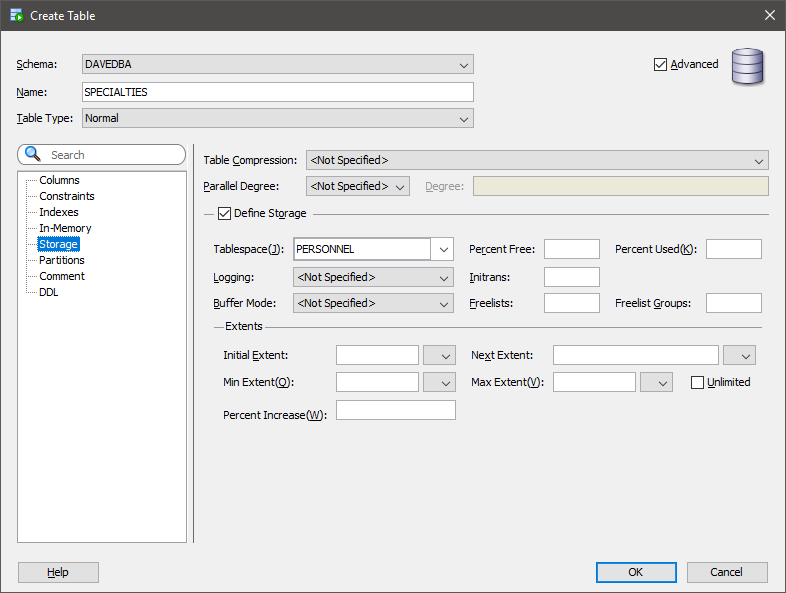
[](https://drive.google.com/open?id=1gJ6XMp0lwEhH-dYYUNej-DDq8JlsyrgU)

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].

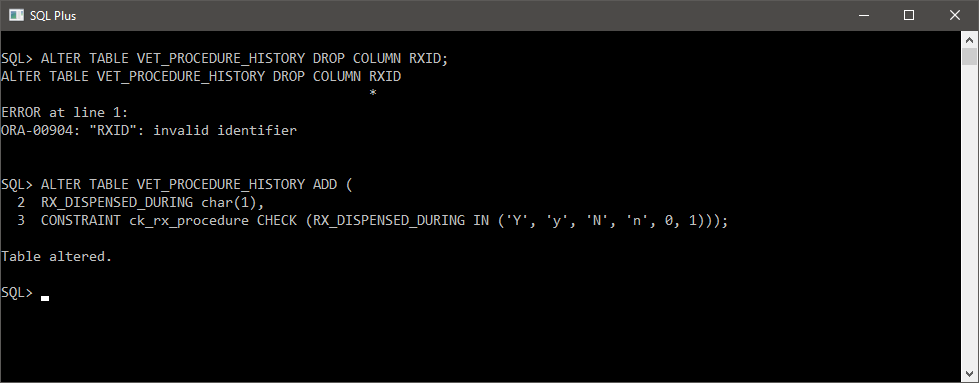


Figure 14

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 [Figure 16 Figure 16].

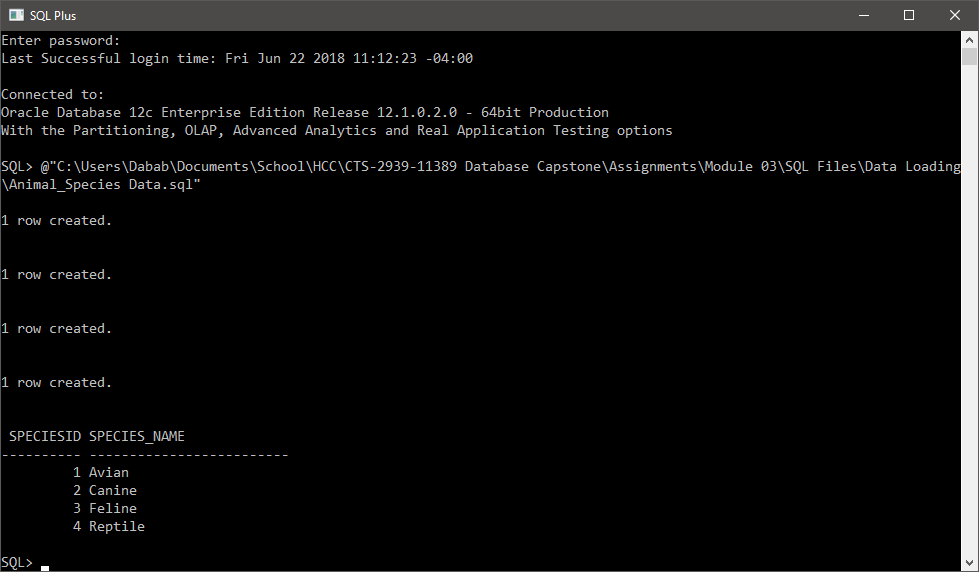
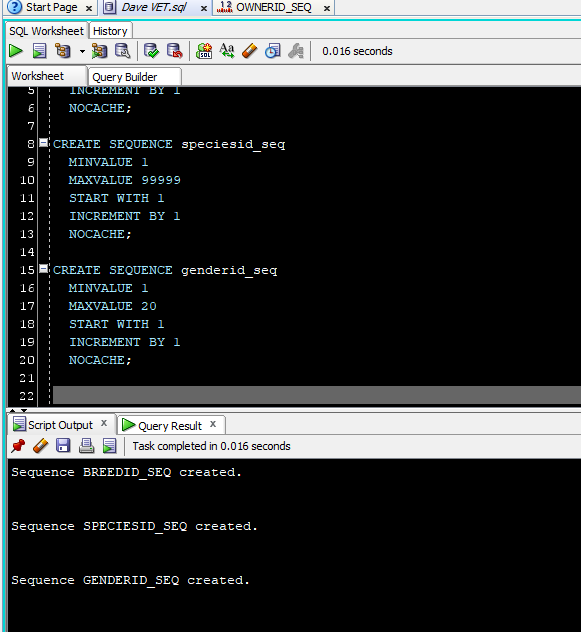


Figure 16

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].

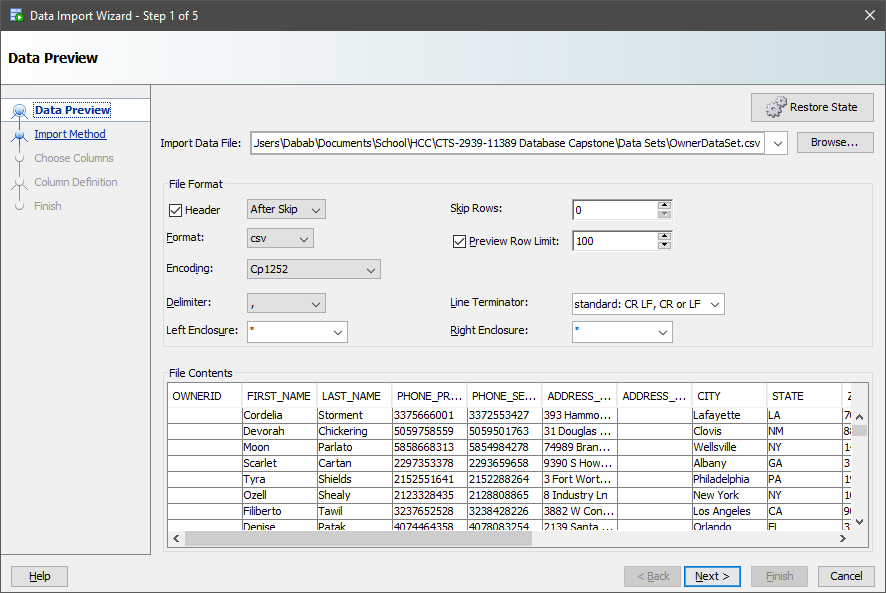


Figure 17

# 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

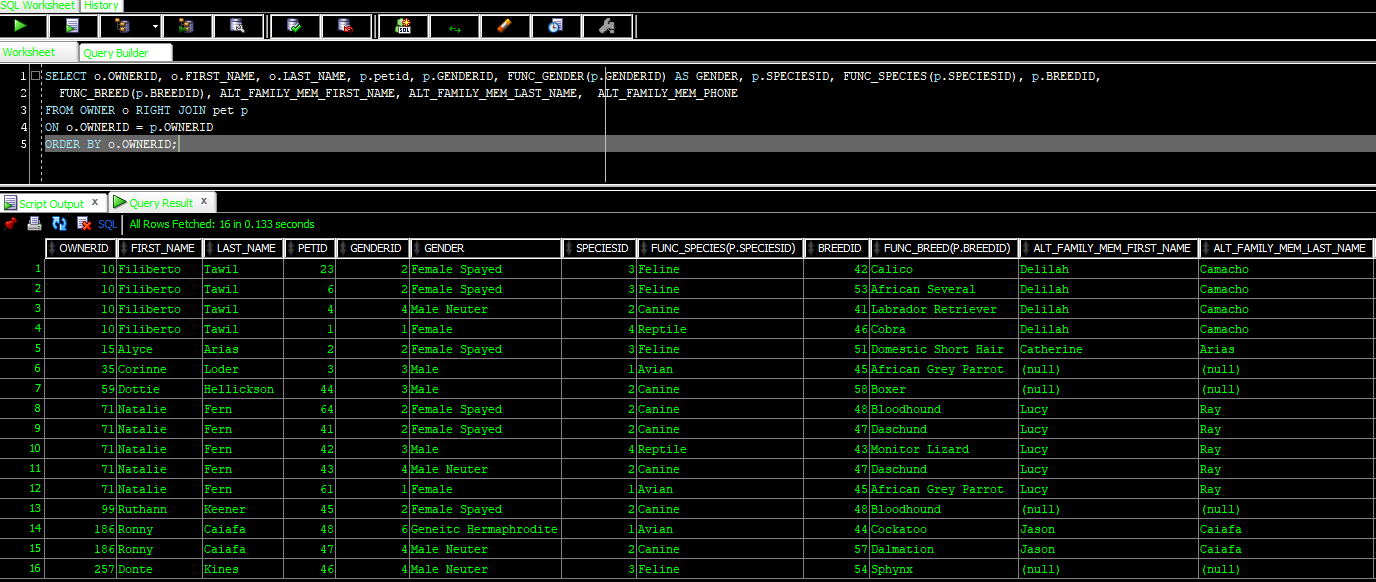
[](https://drive.google.com/open?id=1wfWOHECHH6e4olwVdgrAVC_ua5etJLE0)

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 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].

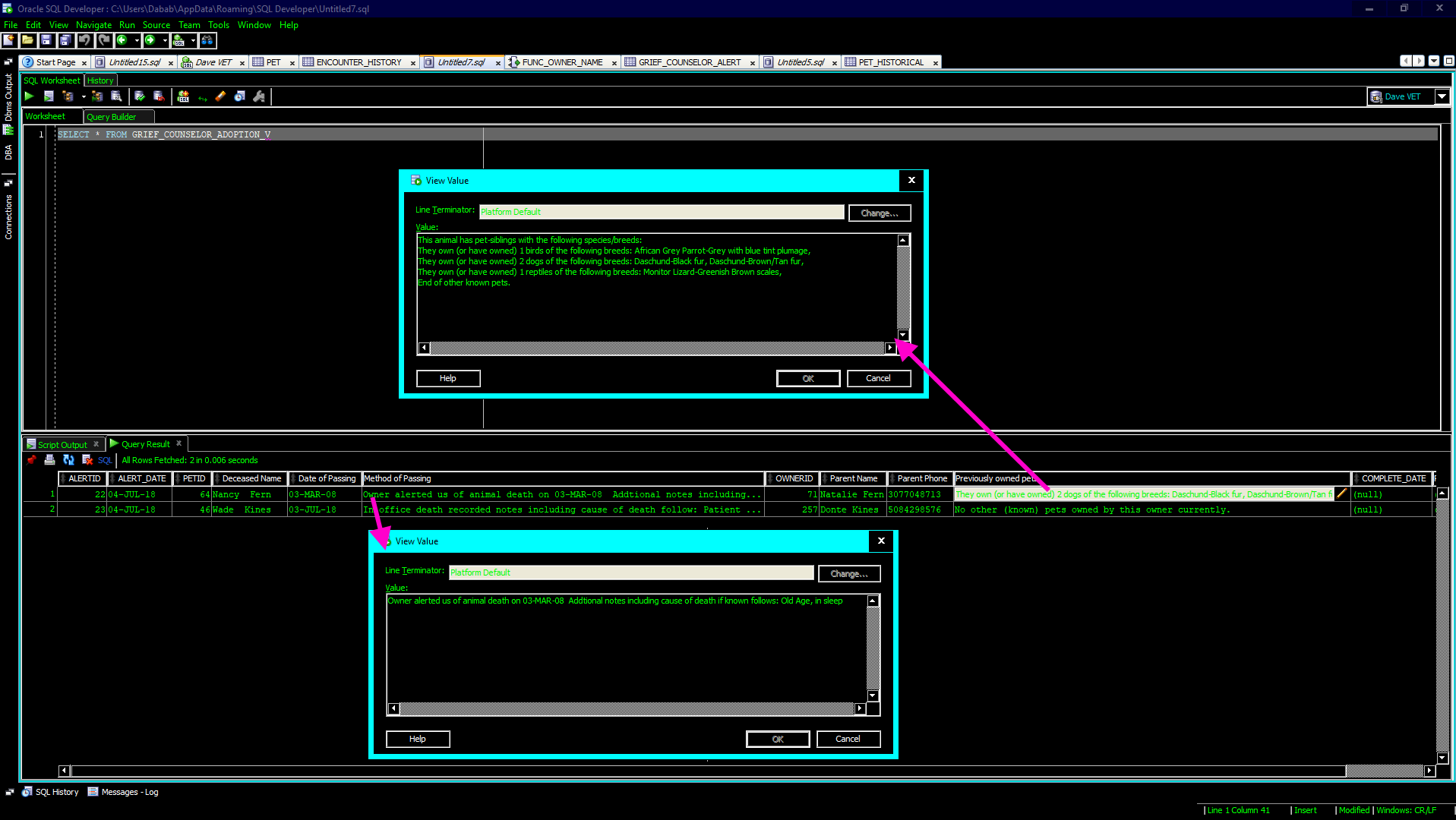
[](https://drive.google.com/open?id=1UKrtqVeD5quCuxUoD8atQ9aFPdtgjT1T)

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.

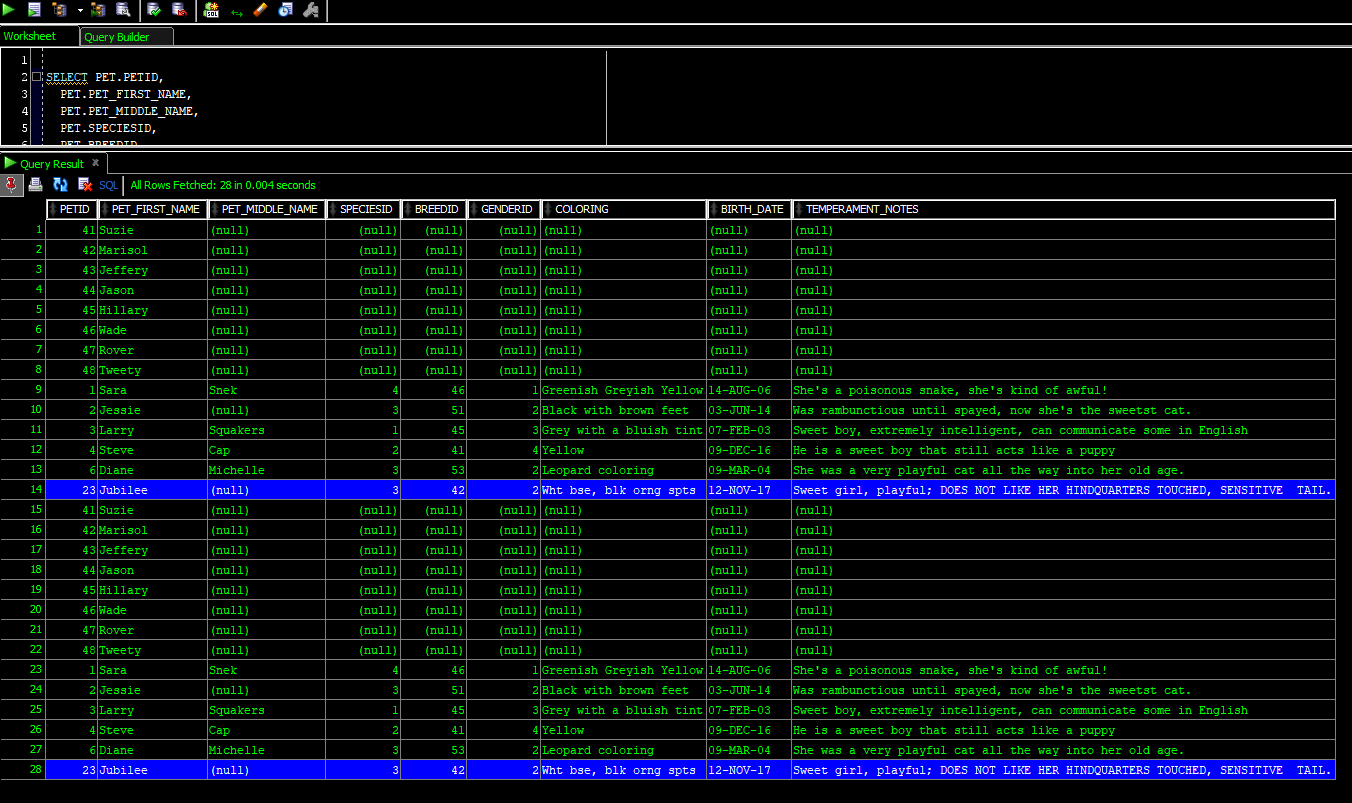
[](https://drive.google.com/open?id=1jXCEWZyuI98eHqWjvehtfuXDCxHWVvUu)

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 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 [Figure 26] 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.

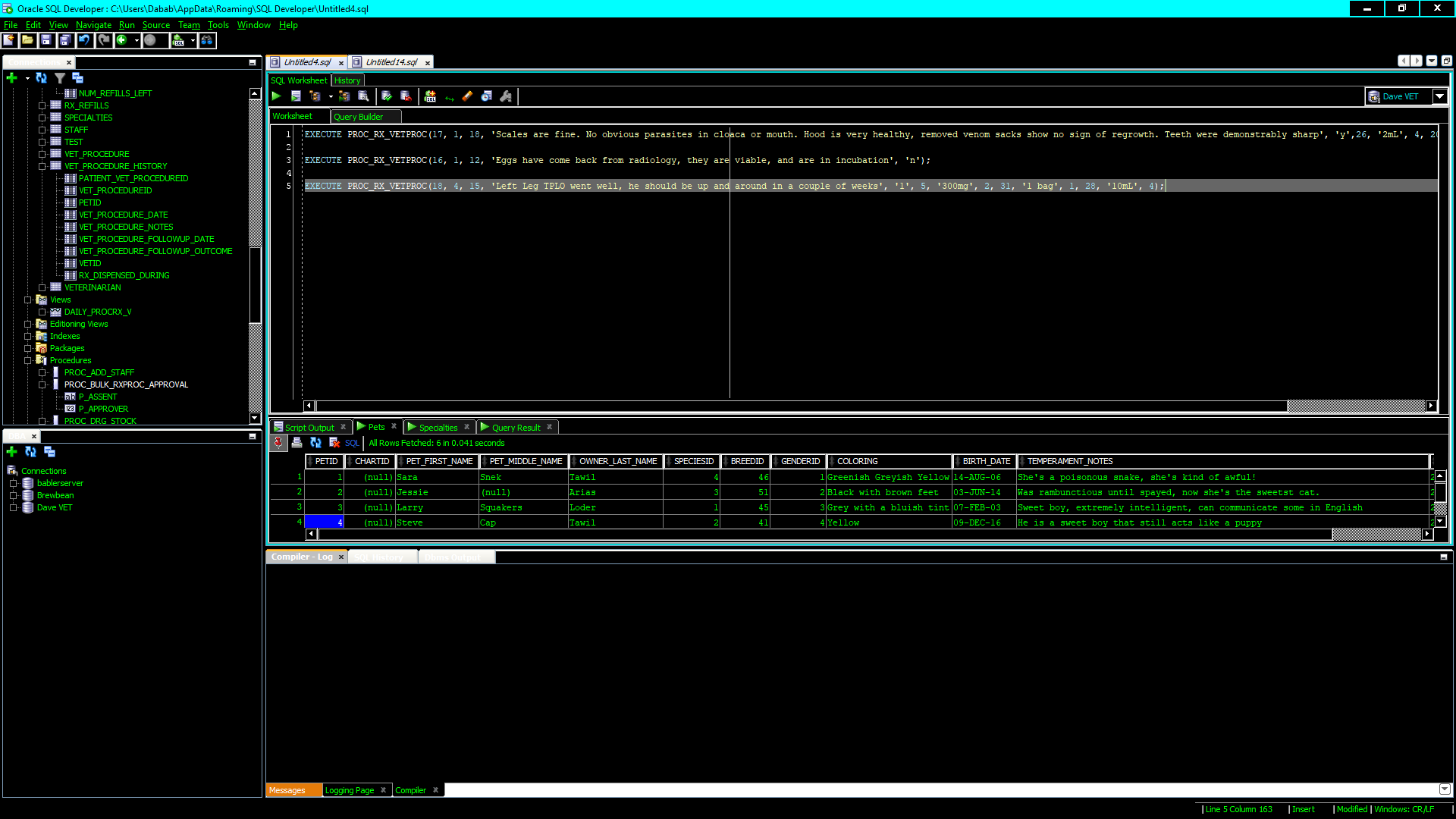
[](https://drive.google.com/open?id=1E6ma2ZdU9wVD1_K-XKPg9Fip5X5jAWEl)

Figure 24

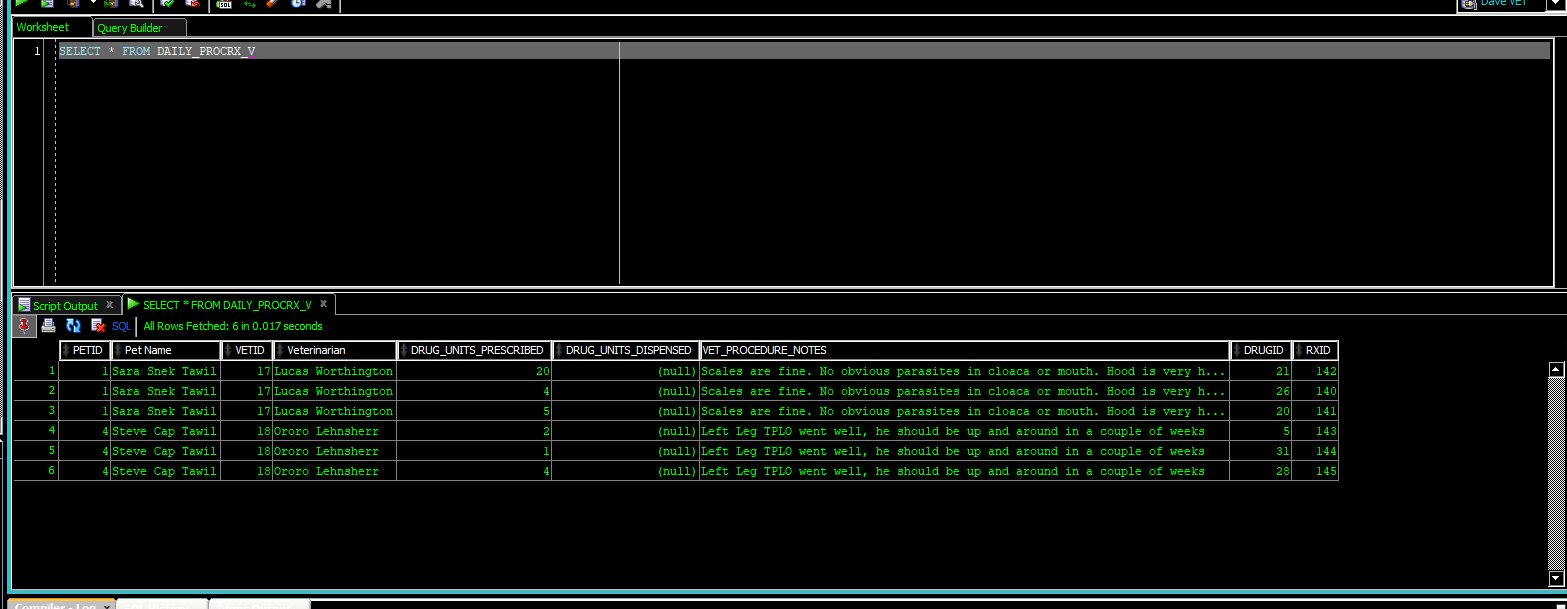
[](https://drive.google.com/open?id=1LQ6gLjQb4HV0lbF7BZfSUNTk10Sro3AD)

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].

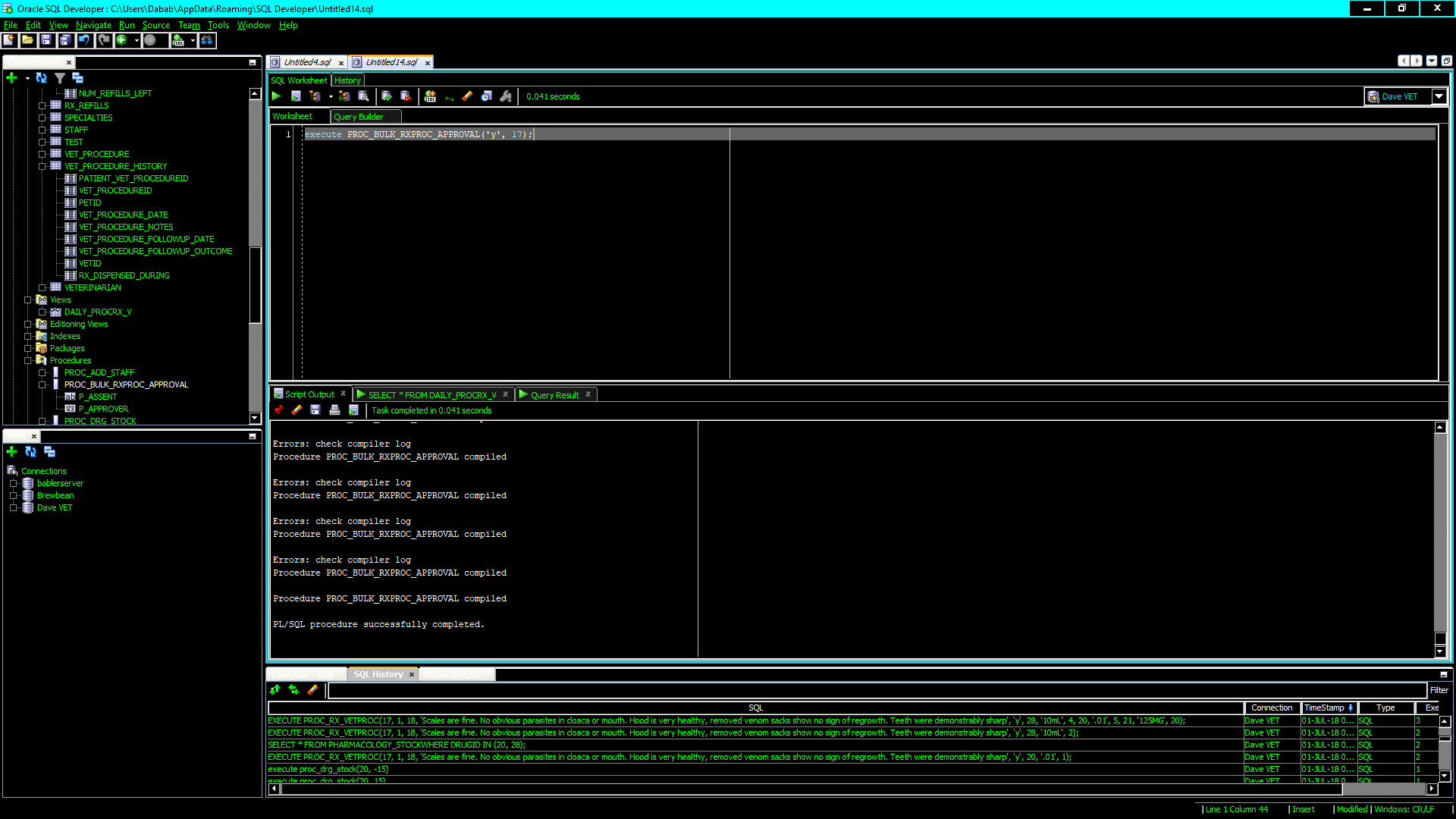
[](https://drive.google.com/open?id=1x5KmKdCKcbbjrN6ti9pIj29_bOx5mdkJ)

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 [Figure 30 Figure 31].

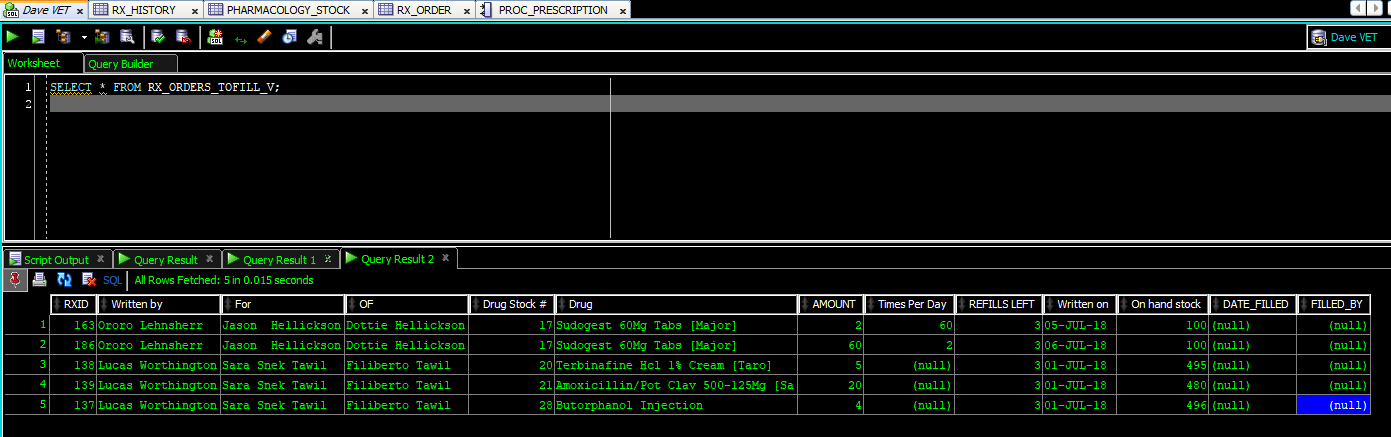
[](https://drive.google.com/open?id=1dK2p9JV6K2xlOmVojdRjAP8B4GcP2Hln)

Figure 30

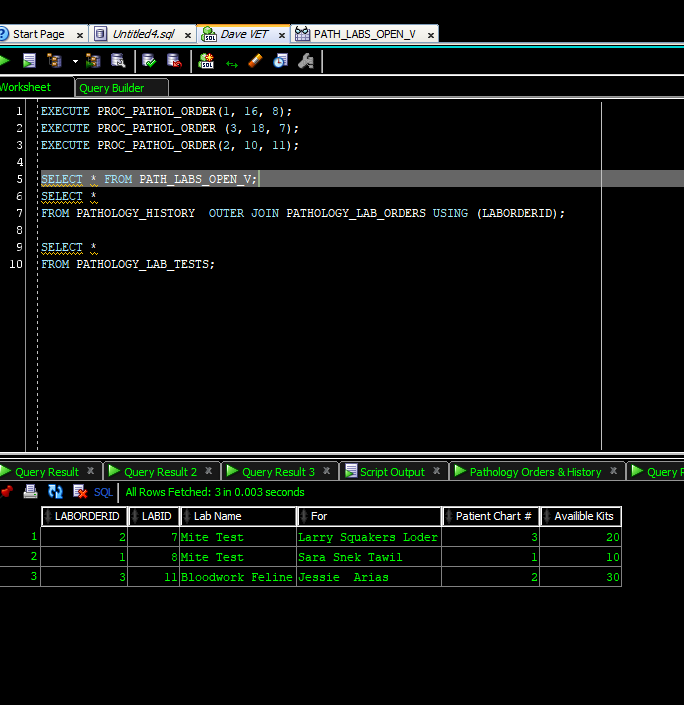
[](https://drive.google.com/open?id=1Y_t2gqY1l4fa_qOggc335uz6LWy7q-Ac)

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].](#_PROC_PRESCRIPTION)
* A procedure for a vet to place a pathology lab order [[PROC\_PATHOL\_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”).
* A procedure to complete a pathology lab order [[PROC\_PATH\_RESULTS]](#_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]](#_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]](#_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.

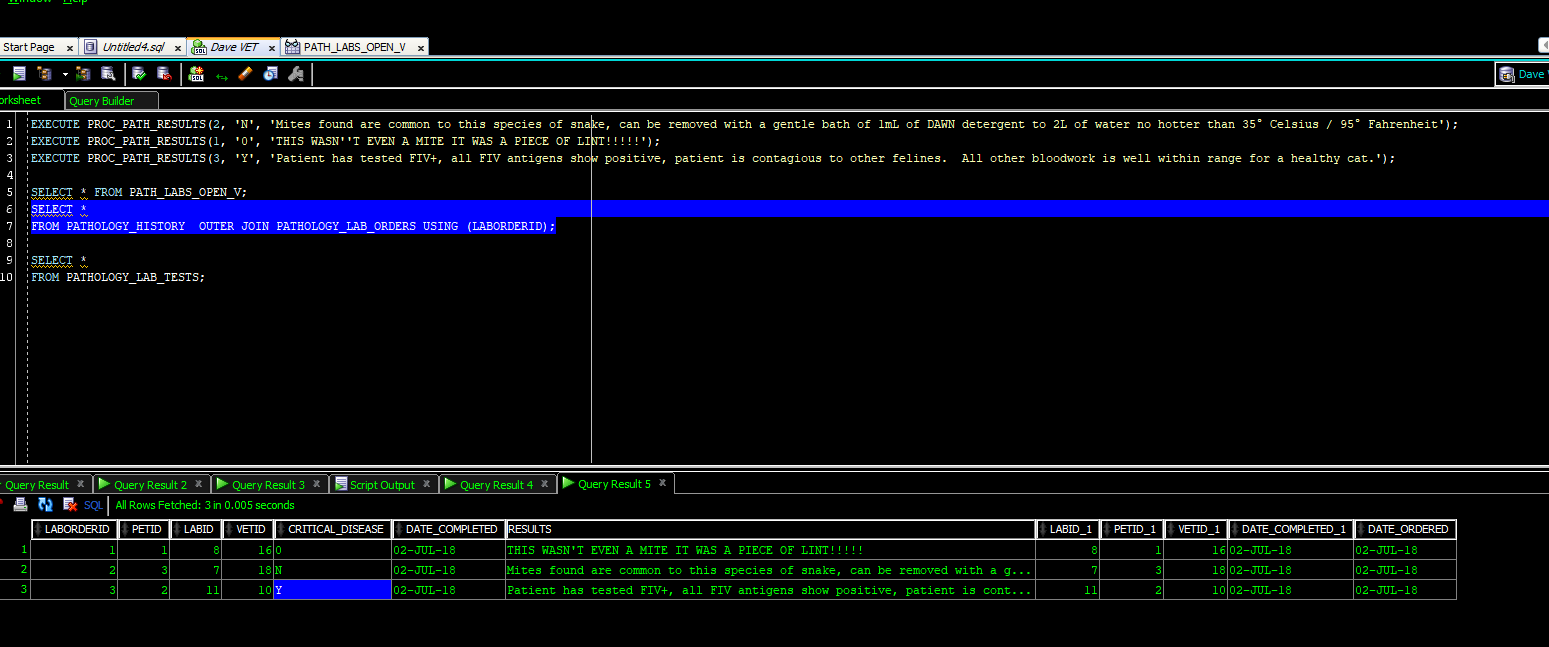
[](https://drive.google.com/open?id=1ny7EPd8O1Xge6vLZdnXpNHtZ6nOw8taz)

Figure 32

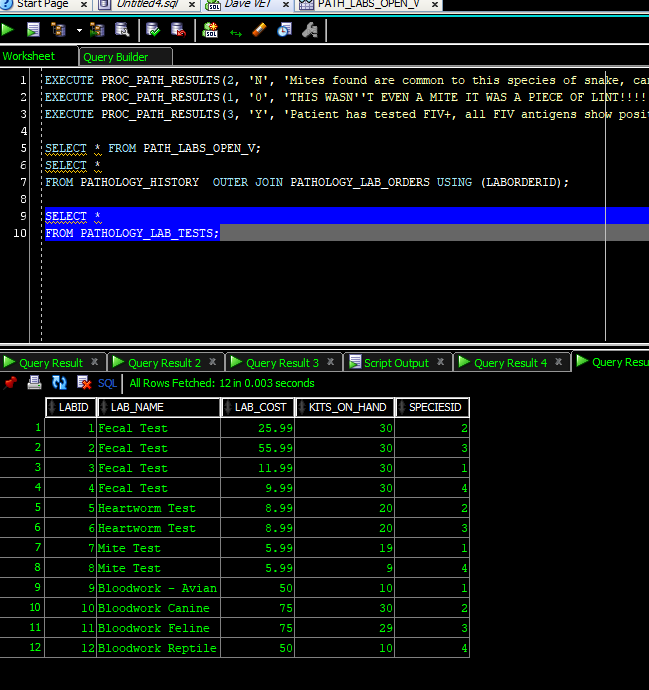


Figure 33

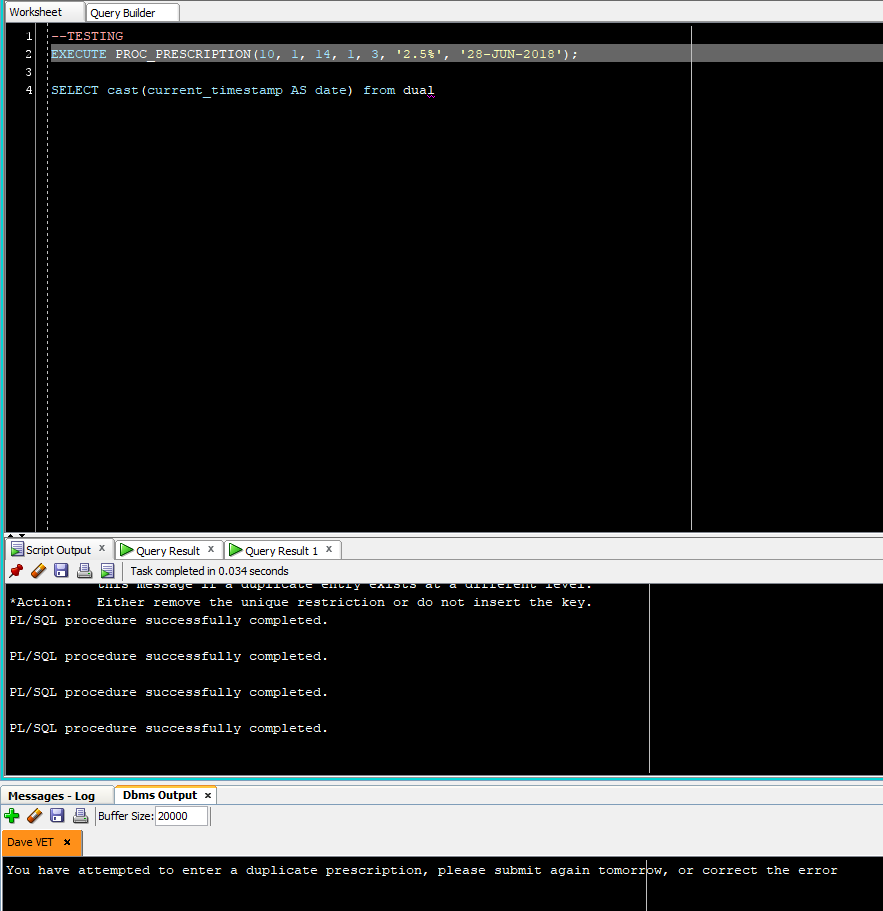


Figure 34

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 36]. Additional values can be added to the check constraints and the program units that call those columns should application-development request it.

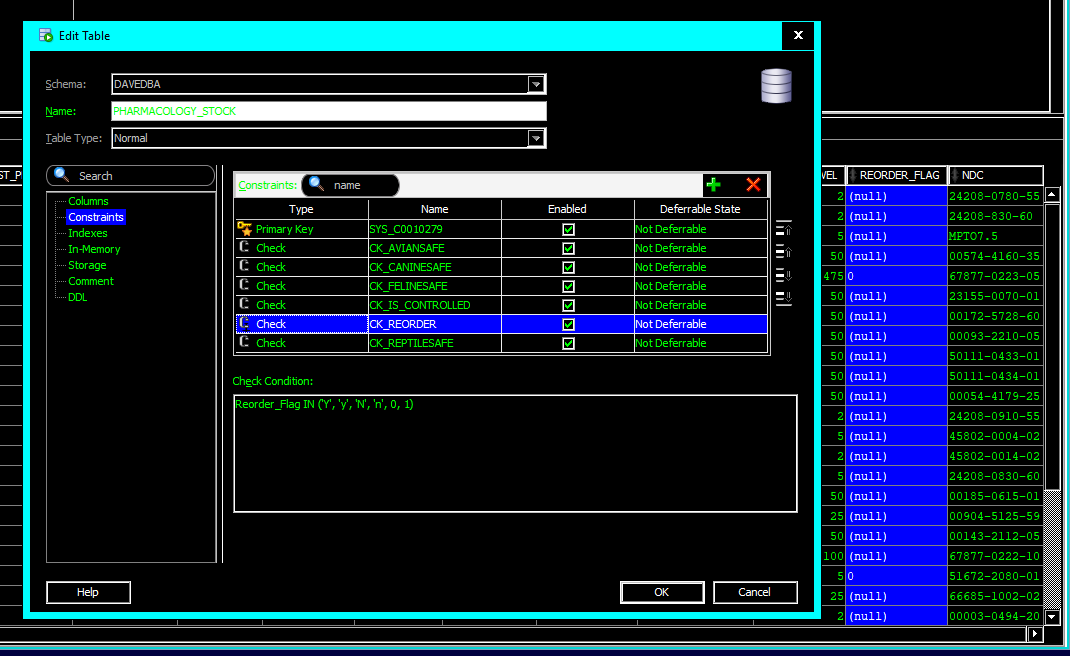
[](https://drive.google.com/open?id=1_fj_hVMh4iVTanRxW6s2cuXZ6wUVMUVZ)

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].

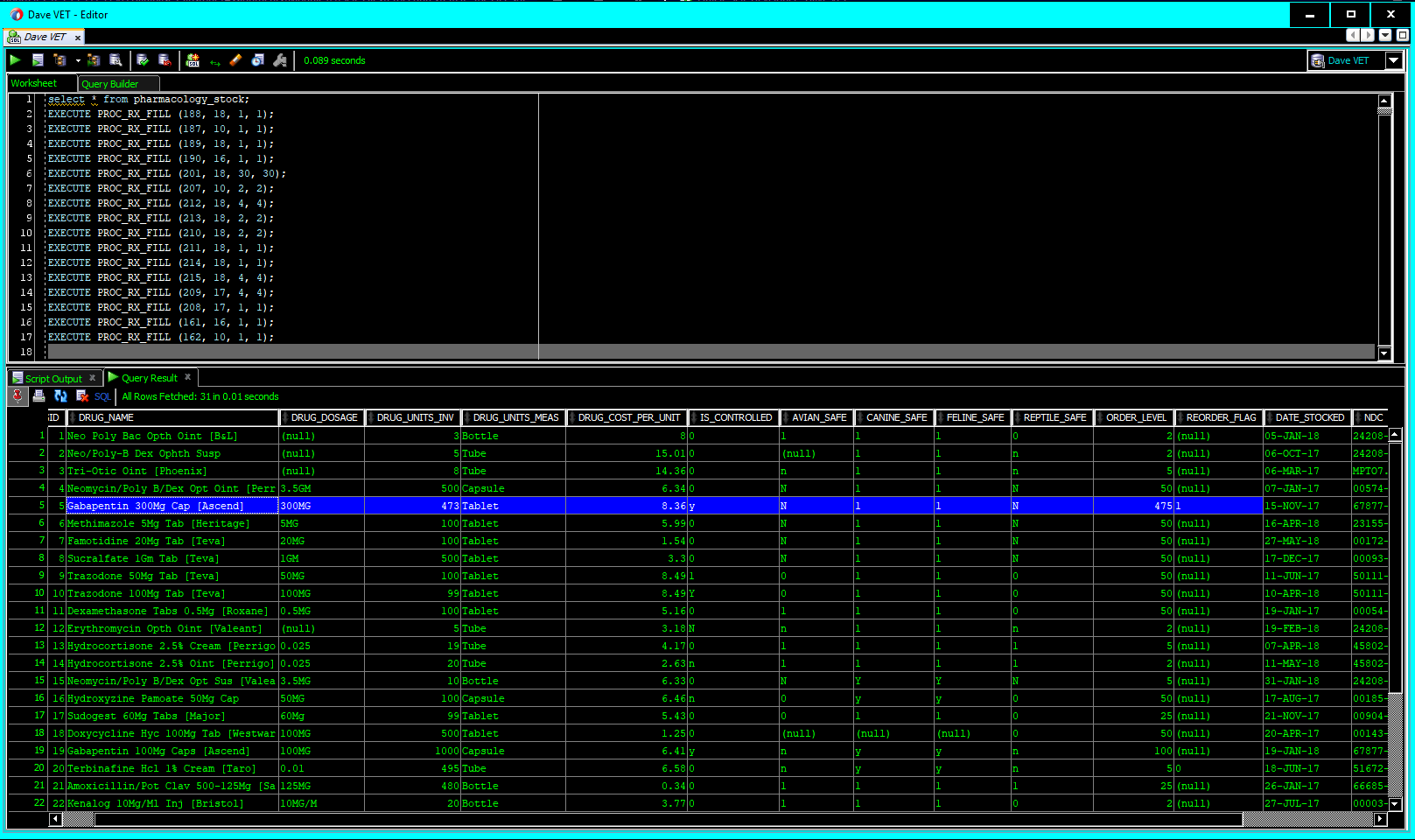
[](https://drive.google.com/open?id=1x3pGkPh2yzmFPvbOlDhXtyorfPGaByOv)

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.

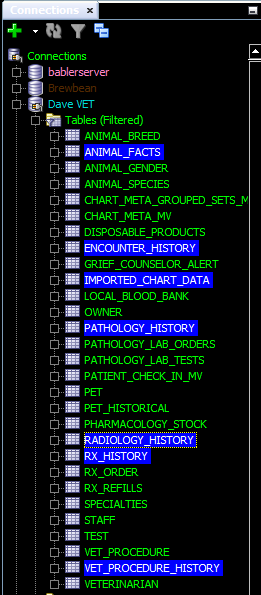
[](https://drive.google.com/open?id=1PFJsMo4d8boj37Gn1yYlkfUWwgVmz4Ii)

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]](#_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]](#_MISC_CHART_VIEWS) 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]](#_MISC_CHART_VIEWS) 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.

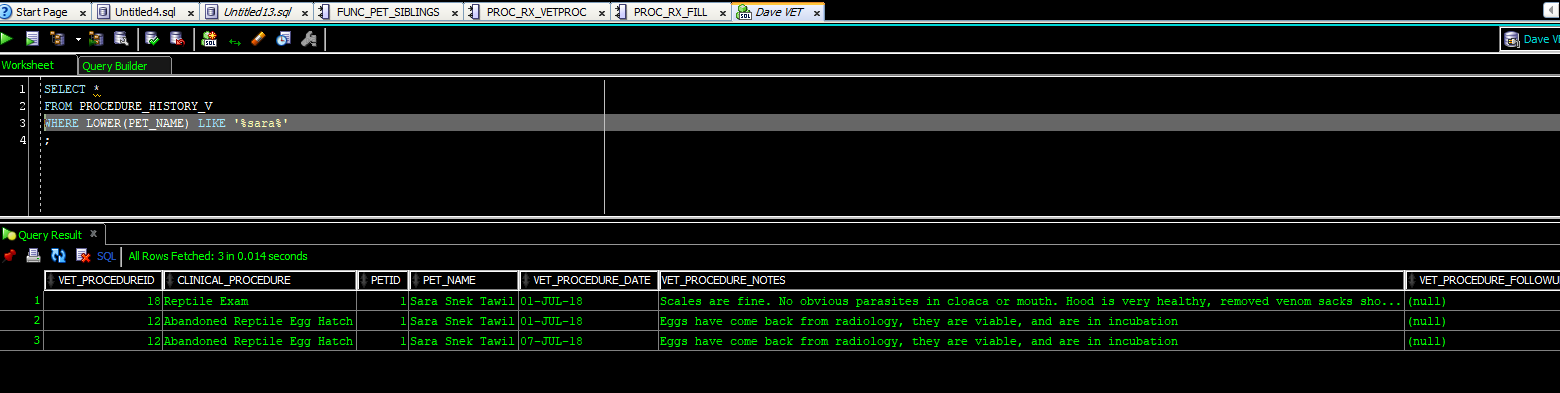
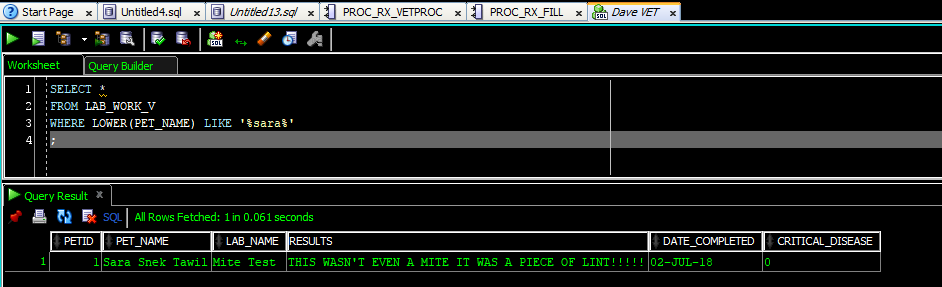
[](https://drive.google.com/open?id=1Yu5pc_QrEAK9P_IS7wp-oBqeclmDKuwB)

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]](#_MISC_CHART_VIEWS) 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 42 shows this view [[RADIOLOGY\_V]](#_MISC_CHART_VIEWS) along with a saved image showing the patient’s X-Ray. Figure 43 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.

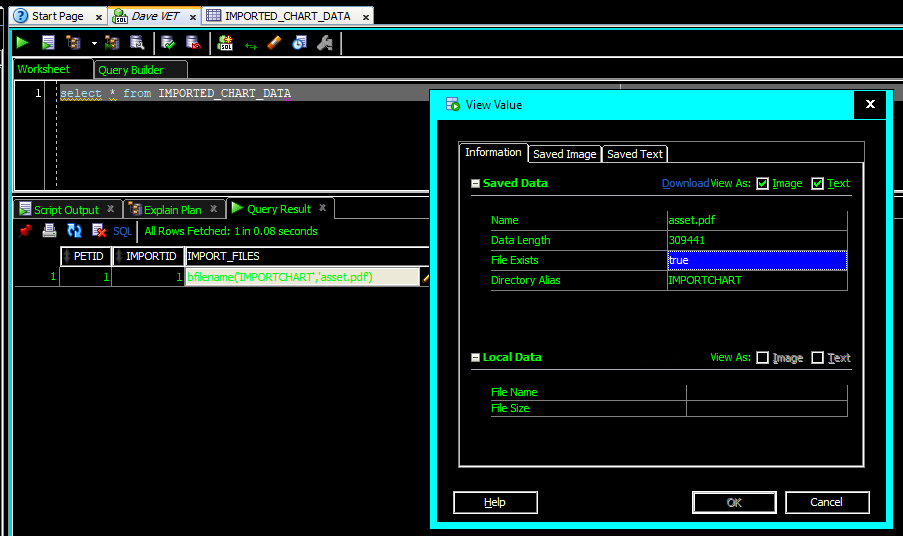
[](https://drive.google.com/open?id=1Zr3UOUnxClZ5jLljl8KvJejszvalrcXv)

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].](#_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.

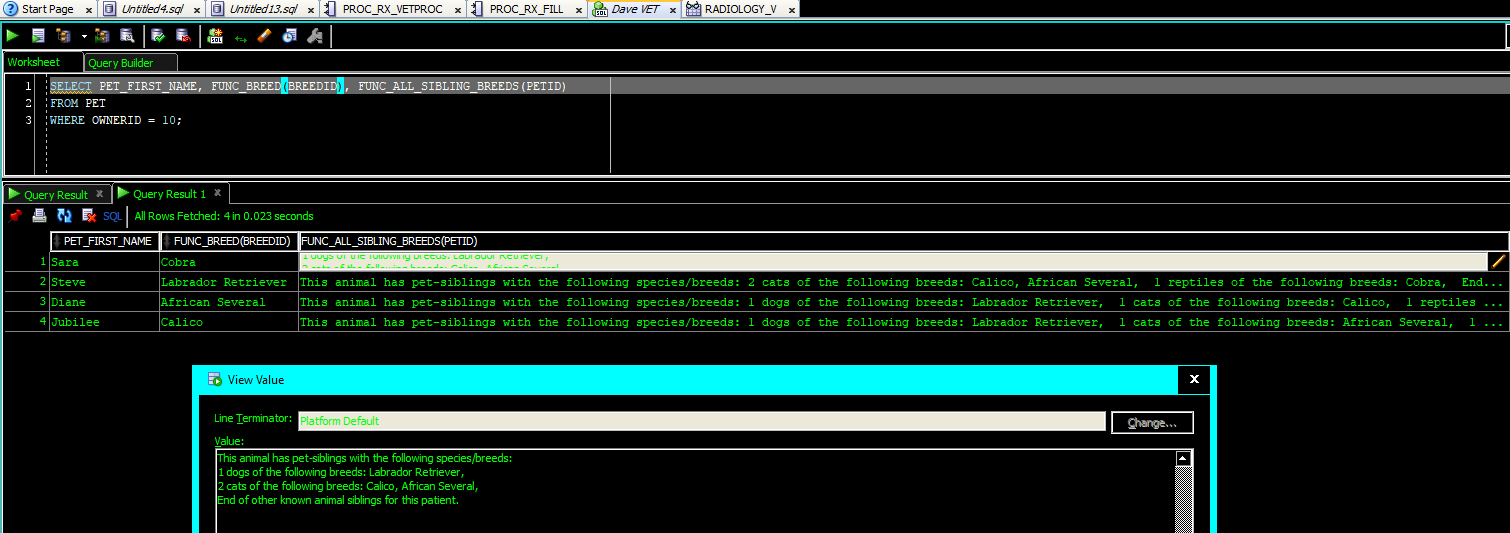
[](https://drive.google.com/open?id=1lv9CYJDSKvj4UkWNzekQxa0uf5NOpTXl)

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 columns must be CLOBs, 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)

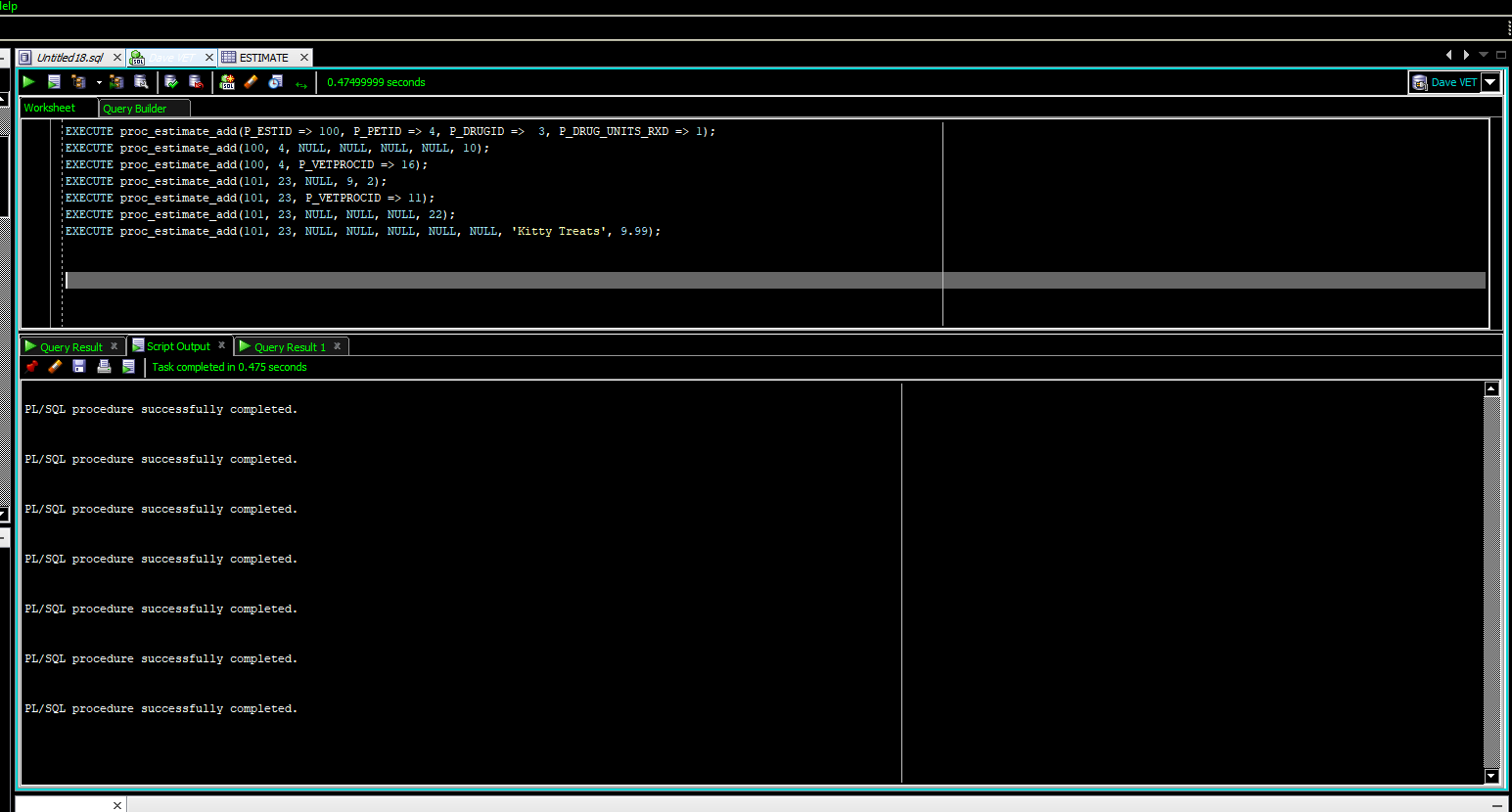
Figure 46

## Invoicing & Estimate Creation

Per CSFDAVD, the invoicing and estimate creation structures were added after the first deliverable was made. This section included in this point of the document to continue the overview of how business rule implementation was accomplished; however, it was completed after work that is discussed later in the document.

### INV-08 Estimate Creation

The basic rules of estimate creation were easily implemented by using the structures of the table and constraints built into [[ESTIMATE TABLE]](#_Estimates_&_Invoicing). A procedure was created to facilitate the opening of an estimate [[PROC\_ESTIMATE\_BUILD].](#_PROC_ESTIMATE_BUILD) First the procedure checks to see if there already is an open estimate for the customer, if true, the procedure outputs the estimate number. If the customer does not yet have an open estimate *for that day* the procedure generates one and outputs the estimate number. Both options for the PROC\_ESTIMATE\_BUILD procedure will add a starter row to the estimate table. This starter row ultimately is used to group the estimate together for an overall sum of services estimated when the view [[ESTIMATE\_V]](#_INVOICE_&_ESTIMATE) is created.

Once an estimate number has been established [[PROC\_ESTIMATE\_ADD]](#_PROC_ESTIMATE_ADD) is used to add items to the estimate [Figure 47].[](https://drive.google.com/open?id=1PdAuN1x4zebd3A-fLZcrgCaNtrHddvJ2)

Figure

Figure 48 shows a copy of the estimate view, which the programmers will use to prepare a printable report.

[](https://drive.google.com/open?id=1mqZl-yYJltlGQyYTzJbU975PlbWqkKVn)

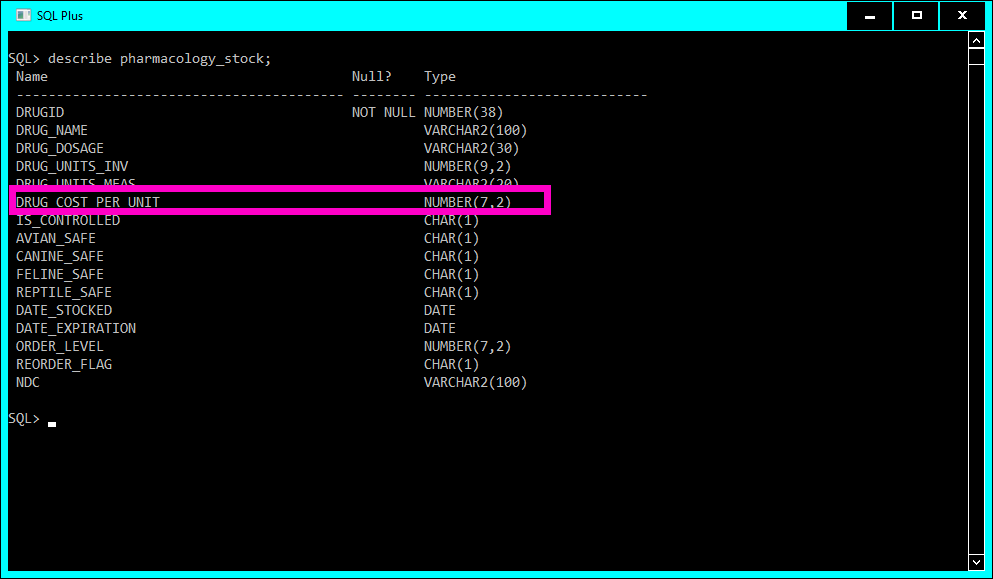
Figure

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

* INV-07: if a customer rejects an estimate it should be deleted from the system.

### INV-10 Invoices Are Done on a Per-Day Basis

It was known during the table creation for other business areas that invoicing support would be added; thus, attributes for costs were added into the data structures. The PHARMACOLOGICAL\_STOCK table is a good example of this [Figure 49]. This was done to set the framework for building invoicing capabilities without having to alter existing data structures in the future.

[](https://drive.google.com/open?id=1P9CbxJnr-aB5kD4AhOKpj5bGVQ0fd_zy)

Figure

Invoices are done on a per-day basis like estimates, so the skeleton for the estimate table was repurposed and modified to create the [[INVOICE\_OPEN]](#_Estimates_&_Invoicing) table. A view was created for invoices that considered INV-02’s rule requirements regarding thirtyday grace periods before late fees [[CUSTOMER\_INV\_V]](#_INVOICE_&_ESTIMATE).

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

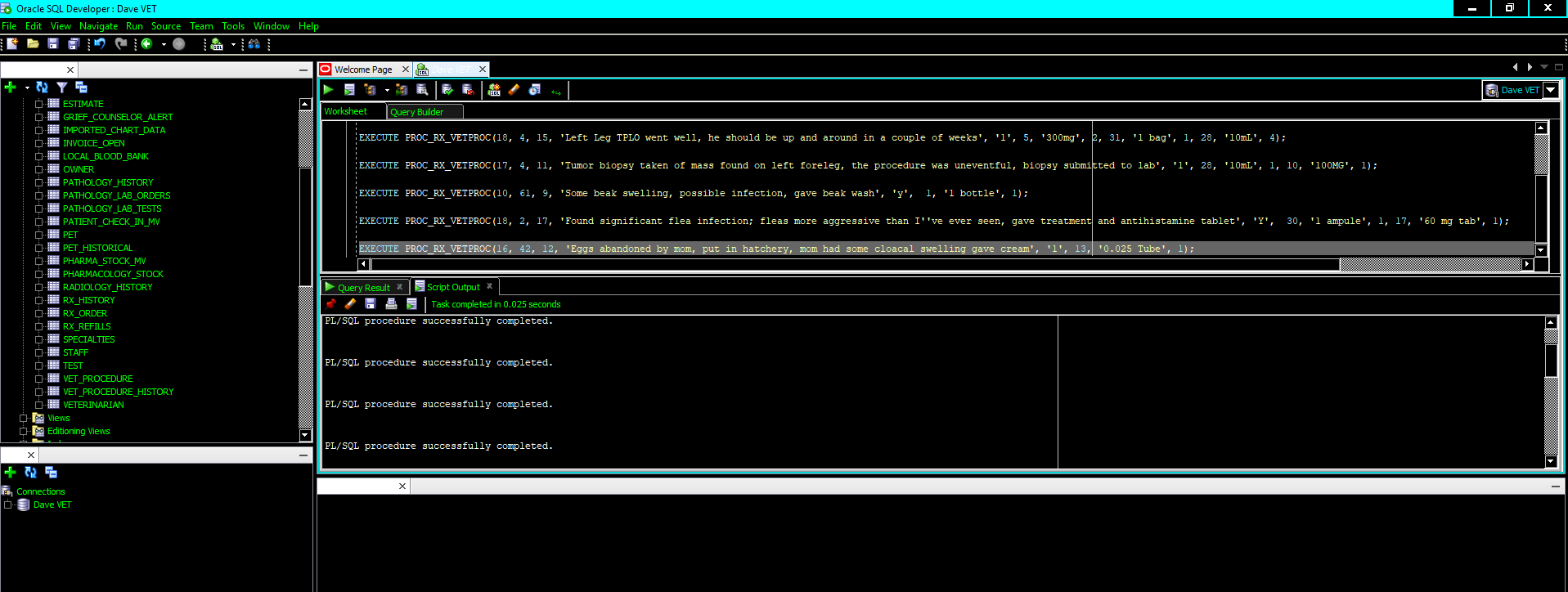
* INV-05: partial payments are not be accepted, invoices are paid in full.
* INV-09: all office visits count as clinical procedures and have charges (also fulfilled during the creation of the CHART tables).

### INV-03 When a Doctor Finishes an Office Visit with a Patient an Invoice Should Generate

Satisfying this rule involved creating procedures to add lines to an invoice and generate invoice numbers that are very similar to their estimate generating counterparts [[PROC\_GETINVNUM]](#_PROC_GETINVNUM) & [[PROC\_ADD\_TO\_INVOICE].](#_PROC_ADD_TO_INVOICE). The business rules specified the invoice update as they entered information into the chart so the various procedures for adding data to a patient chart were modified and the two new invoice generating procedures were embedded in them. To see the updated procedures,[**click here**](#_UPDATED_PROCEDURES_FOR), note: the updated versions have Oracle style commenting in their header denoting the /\*updated for invoicing support\*/.[](https://drive.google.com/open?id=1tK6VsqTF8lqDev72X3EHGK1c7Iv9QilV)

Figure

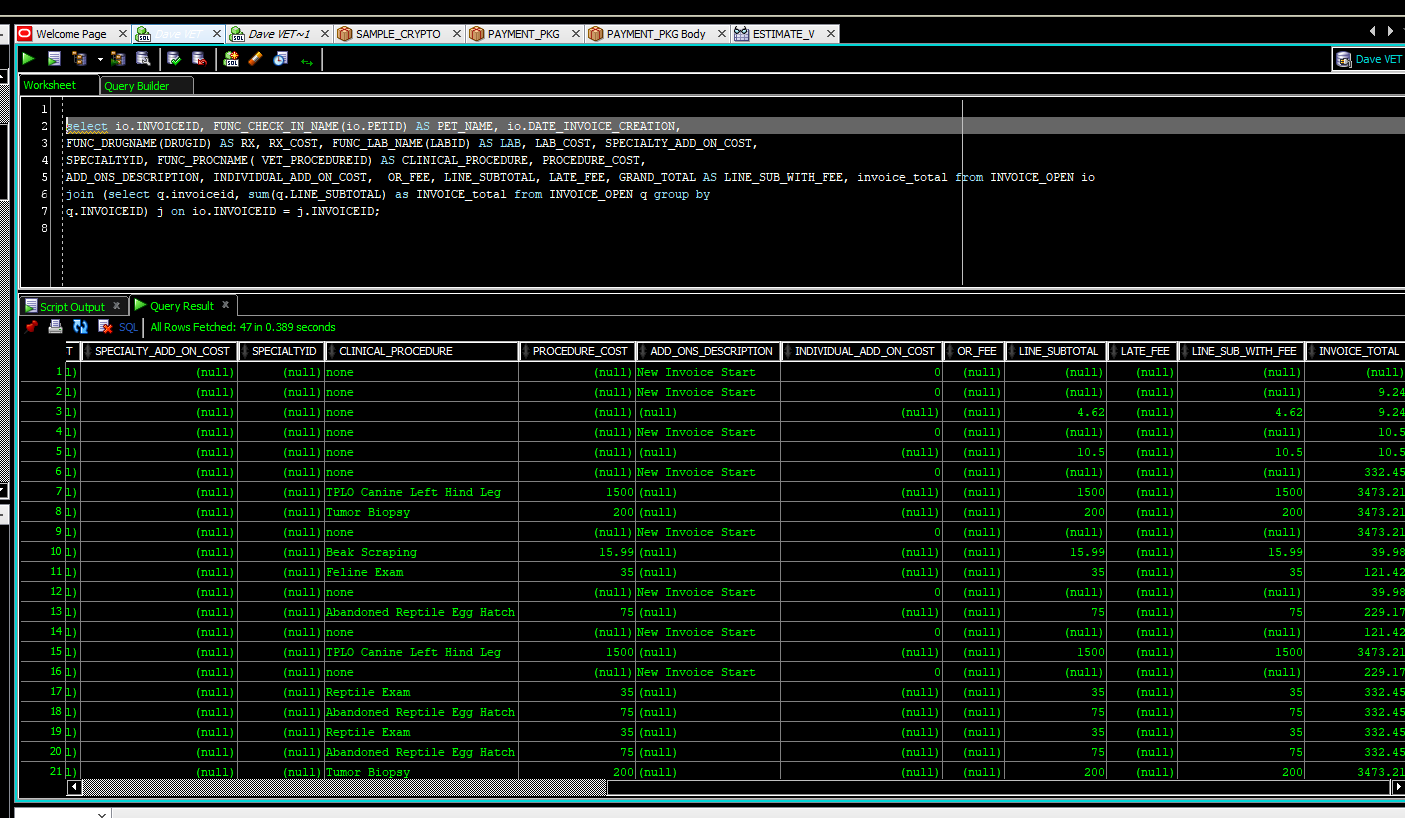
Figure 50 shows the invoice subroutine working upon prescription fill. Figures 51 & 52 respectively show data entry for clinical procedures and then the addition of that to the invoice table.

[](https://drive.google.com/open?id=1_nRCEUDOYAZz81xFGBdFeIGXTH0d1A2c)[](https://drive.google.com/open?id=1jPdIIFcOtpgi2Ob-KK2w7gFdCqkYd_sN)

Figure

Figure

After all the CHART related procedures were added to include the invoicing subroutines the view for invoices could finally be tested

[](https://drive.google.com/open?id=164ktEWLka1TamsTydeScztKMUHpJLN3r)

Figure

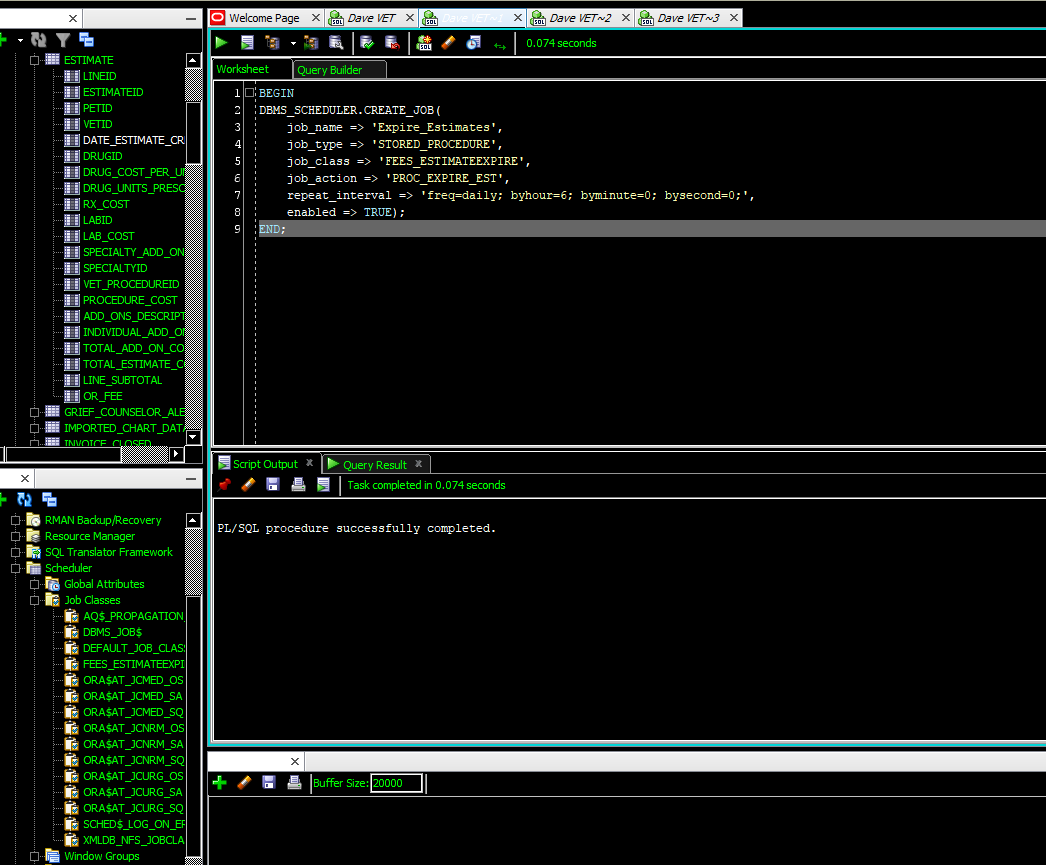
The creation of these procedures also satisfied the following transaction requirements:

* INV-04: costs are added for pathology labs upon lab completion.

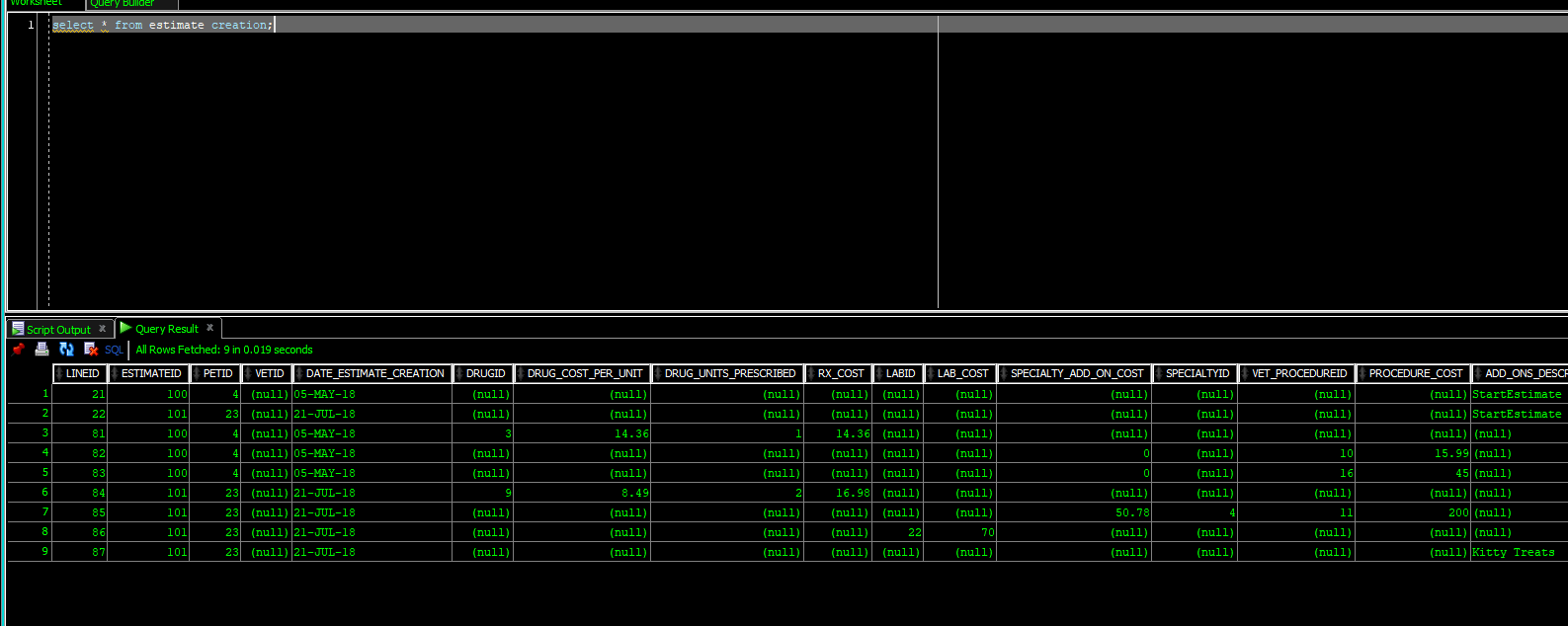
### INV-01 Automatically Add Past Due Fees

The Oracle job scheduler was used to accomplish rules INV-01 which relates to adding automatic 5% late fees to invoices that are 30 days past due, and expiring estimates older than 30 days.

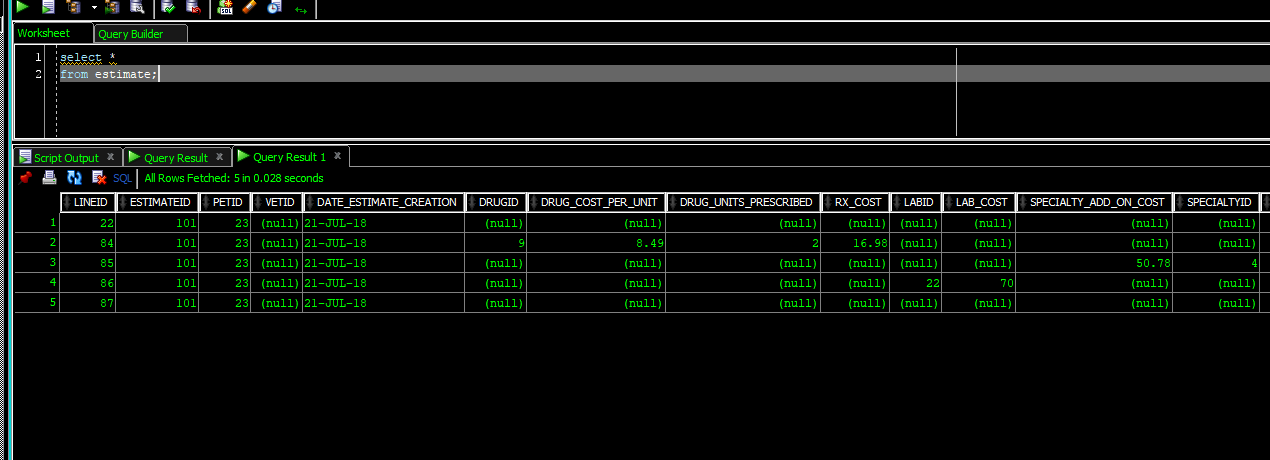
First two procedures were created to manipulate the data in the tables: one to expire old estimates [[PROC\_EXPIRE\_EST]](#_PROC_EXPIRE_EST), and one to add late fees [[PROC\_LATE\_FEE]](#_PROC_LATE_FEE). After the procedures were created a job class was built. A job class is Oracle’s data structure that can be created to logically group similar jobs (Thomas, 2014). [Jobs](#_JOBS) (temporally activated programs) were then created for the procedures [Figure 54].

[](https://drive.google.com/open?id=1jldIWUack8v2nvh2ht0reNOk_cqQuXJf)

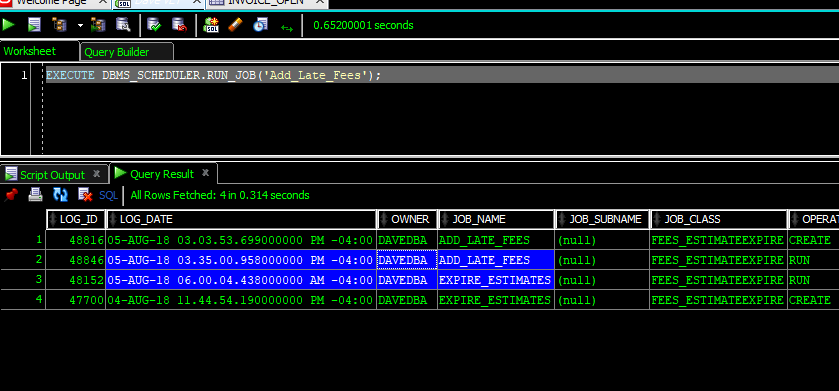
Figure

Figure 55 shows the estimate table pre-job run and Figure 56 shows the expired estimates gone.[](https://drive.google.com/open?id=1jldIWUack8v2nvh2ht0reNOk_cqQuXJf)

Figure

[](https://drive.google.com/open?id=1mLPzklsT5zCS3w29NdTWITSu0C2mJeXO)

Figure

The data dictionary view ALL\_SCHEDULER\_JOB\_LOG shows that the estimate creation job was run automatically while the late fee job was run manually [Figure 57].

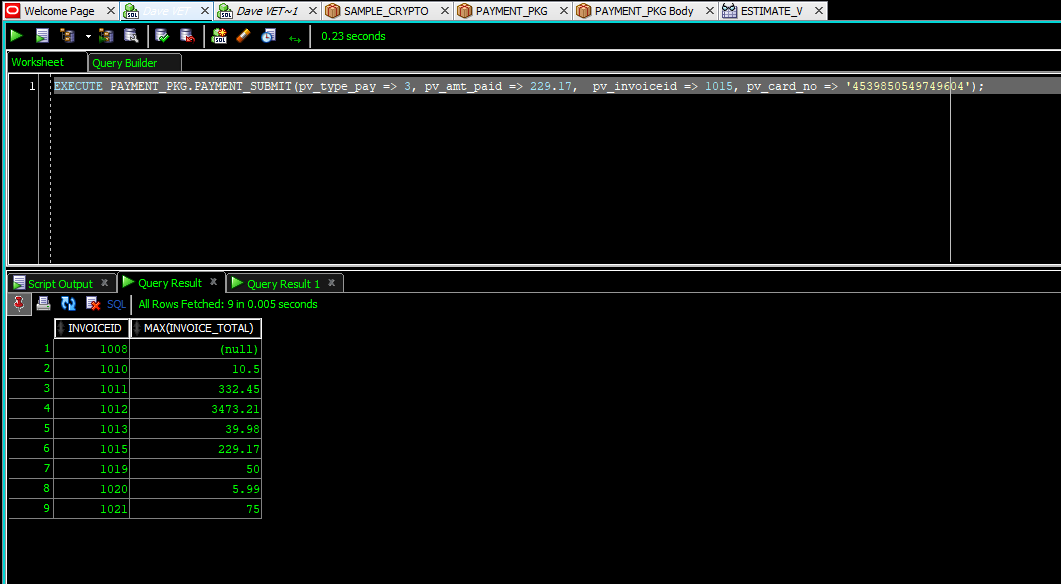
Figure

## Out-of-Scope Request: Encryption and Payment Support

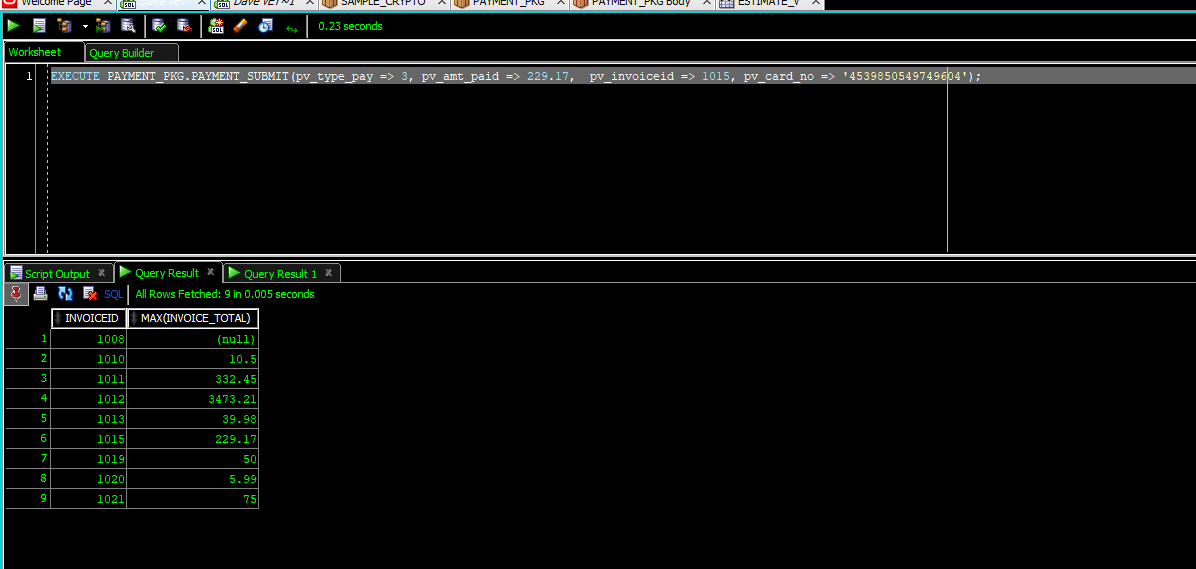
The veterinarians approached the development team concerned with the efficiency of their current payment tracking system. The stakeholders asked if encryption for creditcards and a form of payment tracking could be incorporated into the system, “just in case”. Though the request fell out of scope with the original project the development team was happy to accommodate the concerns of the stakeholders. This section discusses the creation of the package [[PAYMENT\_PKG]](#_PAYMENT_PKG).

### OOSR-01 Payment Submission

The PAYMENT\_SUBMIT subroutine of the PAYMENT\_PKG allows for an associated invoice to be closed out. Values are entered in the procedure for the type of payment, the invoice id, and the amount paid. The optional constraints (using the DEFAULT NULL tag) for credit card or check information were also added to the PAYMENT\_SUBMIT procedure. Figure 58 shows an ad-hoc query of the INVOICE\_OPEN table with distinct totals and invoiceids, this is before execution of PAYMENT\_PKG.PAYMENT\_SUBMIT. After a successful payment the invoice is removed from INVOICE\_OPEN and thus no longer shows in the ad-hoc query [Figure 59].

[](https://drive.google.com/open?id=1IxfbS4OrjPXR5VFycj_TpwU51Ko1CLy1)

Figure

[](https://drive.google.com/open?id=11QHJlgOwi29G6xaYTmudmyDdT3Ik4ond)

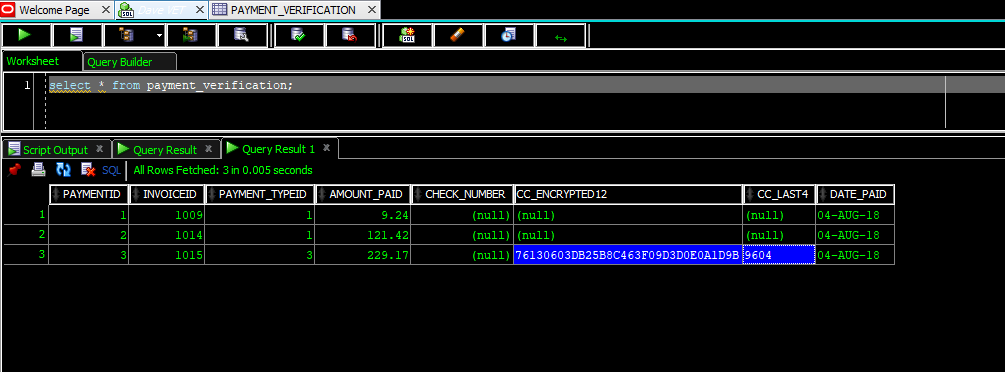
Figure

#### OOSR-02 Closing an Invoice

Figure 59 shows the removal of the invoice. This is accomplished by the subroutine INVOICE\_MOVE of the [[PAYMENT\_PKG]](#_PAYMENT_PKG) package, which is called by the PAYMENT\_SUBMIT procedure. INVOICE\_MOVE allows for a paid invoice to be cut out of the INVOICE\_OPEN table and moved to the INVOICE\_CLOSED table.

#### OOSR-03 Encryption of Credit CArd data

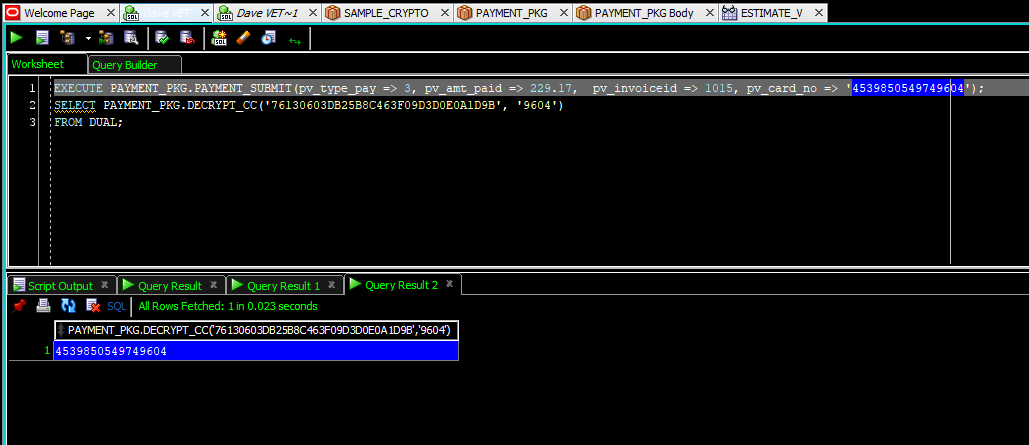
Traditionally creditcard data is encrypted up through the 12th digit and digits thirteen through sixteen are left unencrypted for tracking and payment verification purposes. This standard for encryption was included when the Encryption and Decryption subroutines were made. ENCRYPT\_CC function is called by the PAYMENT\_SUBMIT procedure of the [[PAYMENT\_PKG](#_PAYMENT_PKG)] package. This function takes character data and converts it to RAW (hexadecimal) data and prepares encrypts it using one of Oracle’s included encryption algorithms, the PAYMENT\_SUBMIT procedure then fills in the PAYMENT\_VERFICATION table with the encrypted data [Figure 60].

[](https://drive.google.com/open?id=12EK9xOSQqKTSv6XiP1dkft5j9zCXwhxQ)

Figure

### OOSR-04 Decryption of Credit Card Data

For chargebacks, credit card disputes, and auditing purposes the accountant may need to decrypt the creditcard data. A function DECRYPT\_CC was wrapped in the [[PAYMENT\_PKG](#_PAYMENT_PKG)] package to accomplish this. The raw encrypted part of the credit card along with the unencrypted last four digits are passed into the function and the original card number is returned [Figure 61].

[](https://drive.google.com/open?id=1jQqBmOsPvt3cVxZ_D-leuI6mph-jOAY0)

Figure

## 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]](#_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].

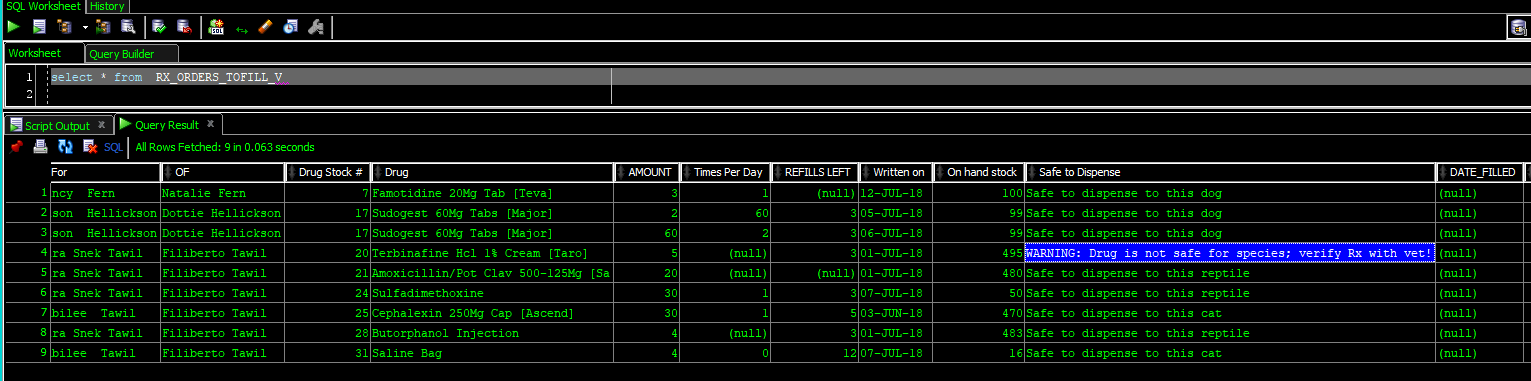
[](https://drive.google.com/open?id=15-ESgXvhreuTGX6XyOgrRJhgOKzECS72)

Figure 62

### 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 63

### 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],](#_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 64

Then a function was created to make the header more natural for a user to read. The function is generated by the PETID [Figure 50].

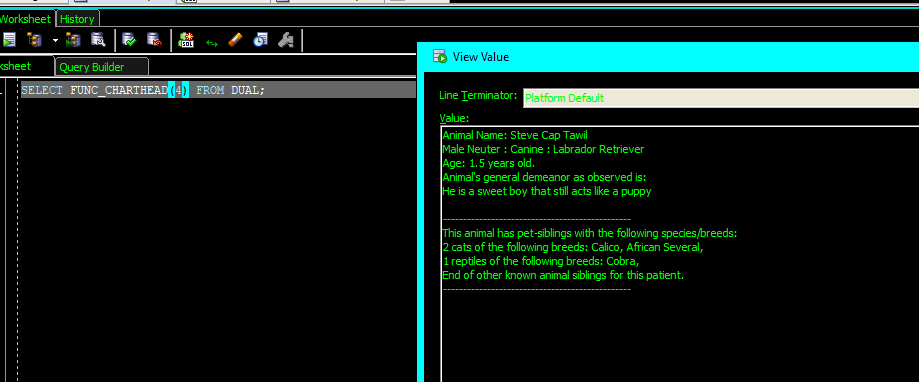
[](https://drive.google.com/open?id=1ibz1pnGixbAJsIorg0SENOHL9p9gsBbJ)

Figure 65

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

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

Figure 66

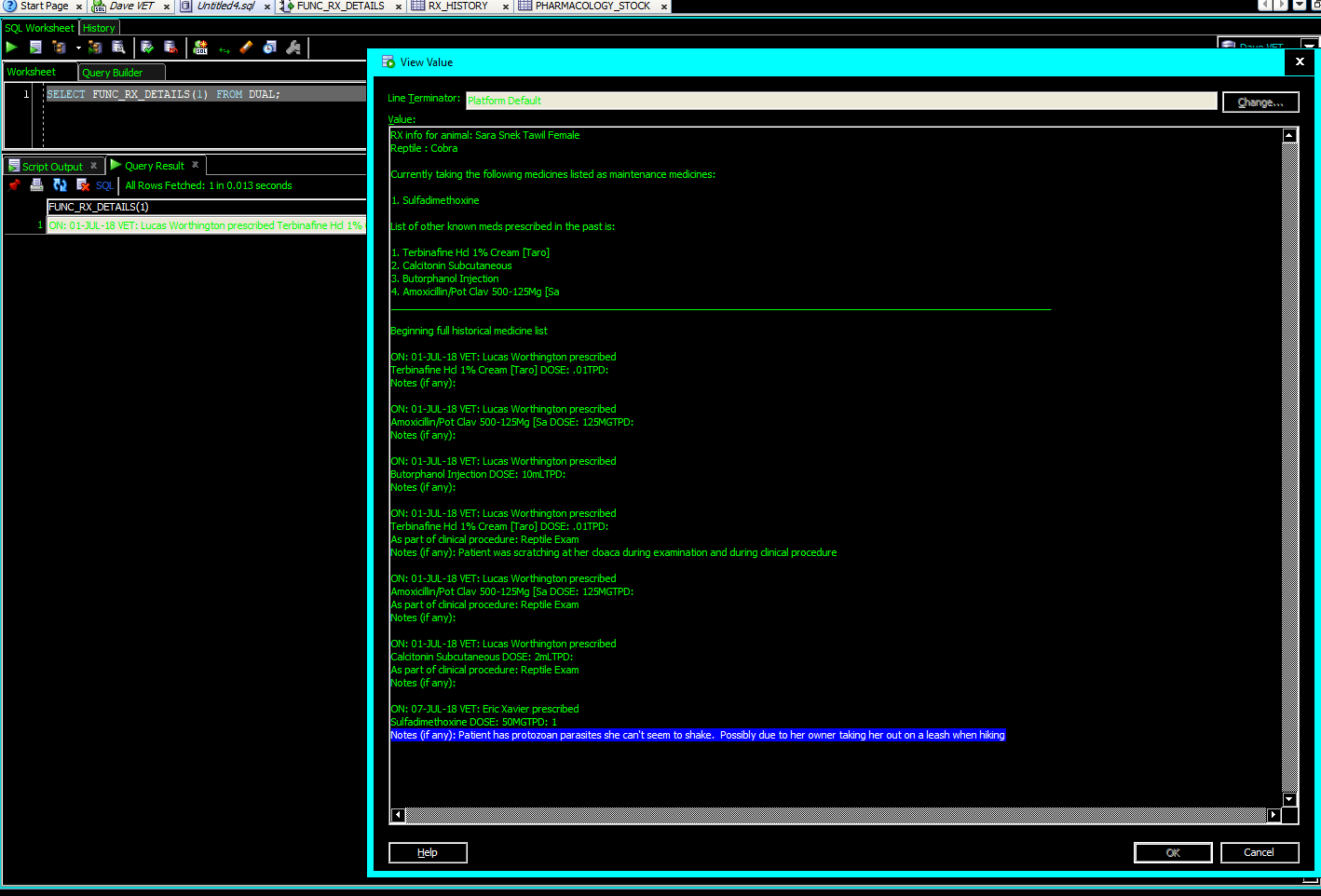
[](https://drive.google.com/file/d/1-K-Rqak91GaH8LuHYJwp5VTJn_i1JF5N/view?usp=sharing)

Figure 67

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

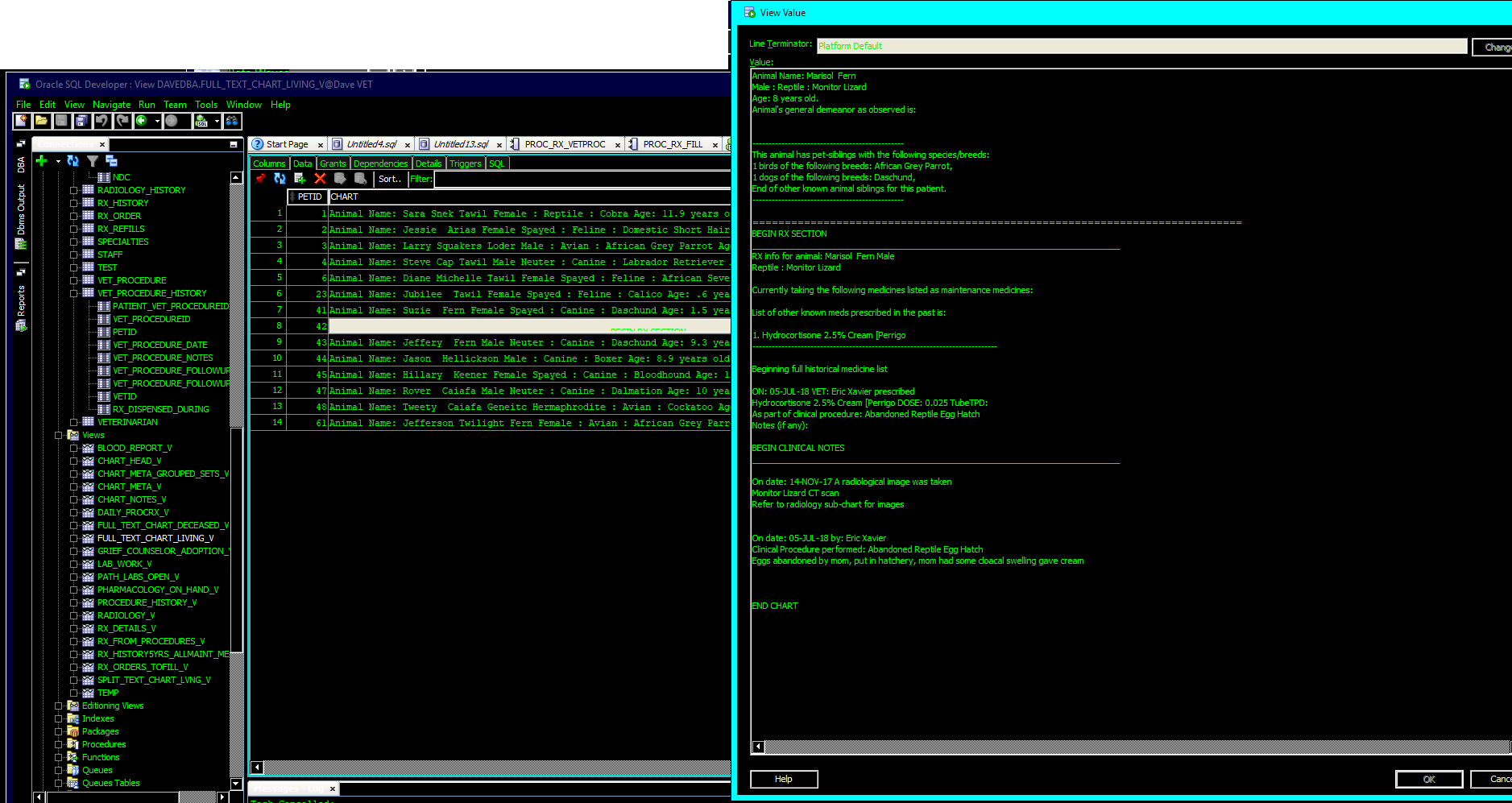
[](https://drive.google.com/open?id=1YpYCKdNYgHfGfY9cIkkyn0rBAOHlANYr)

Figure 68

All these functions and procedures were wrapped in an Oracle Package [[CHART\_PKG]](#_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 69

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]](#_CHART_META_MV_&_cHART_META_GROUPED_) which feeds [[CHART\_META\_V]](#_CHART_META_V_&_CHART_META_GROUPED_S).

### 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 queries 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 70

#### Search procedure for the traditionally JOINed chart

A search procedure was also created for querying the traditionally JOINED chart and allows for using the condition ‘LIKE’ which creates a pseudo-fuzzy search capability. The procedure [[PROC\_CHART\_SEARCH]](#_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.

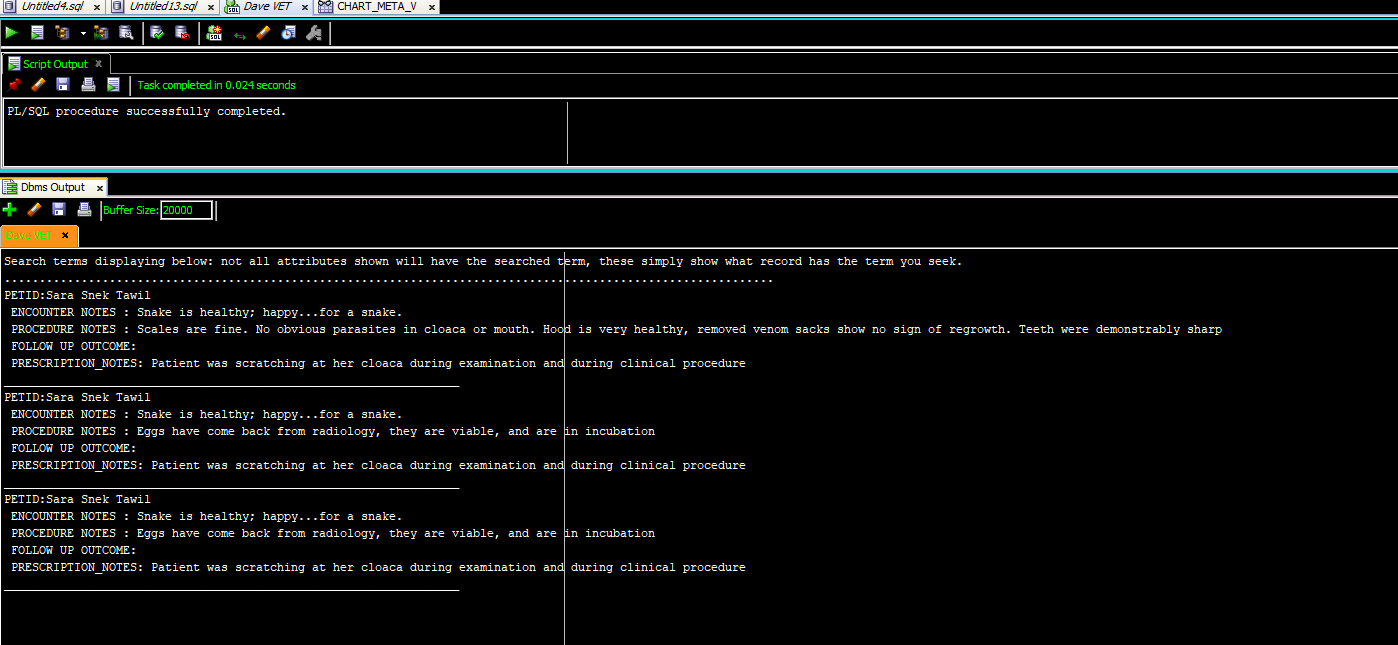
[](https://drive.google.com/open?id=1nhKTvTqCNs8jD2O2Gi95bK_yQqNRPBwK)

Figure 71

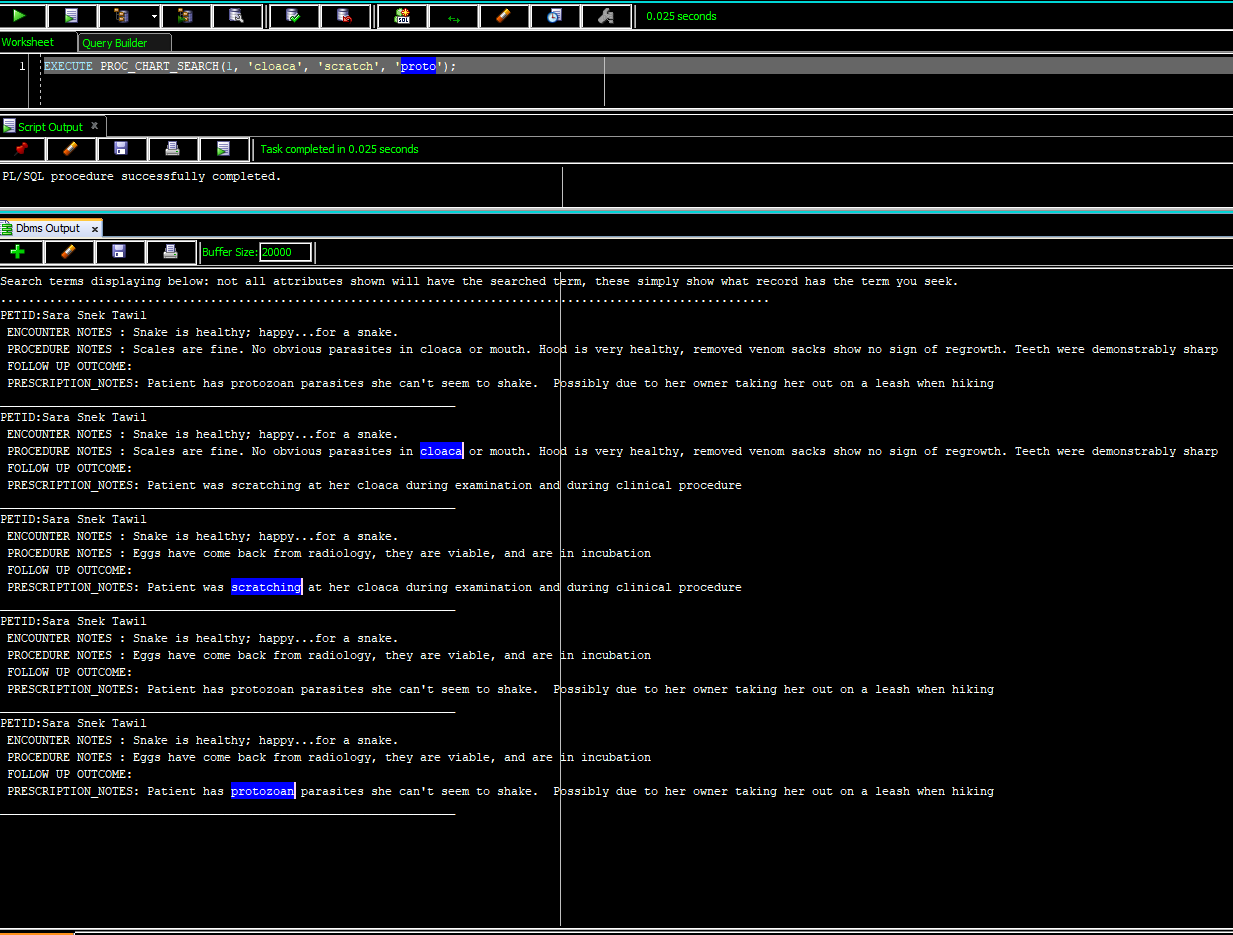
[](https://drive.google.com/open?id=1MWxFKnpUaFmsryG5upIrjqfuQq5ZBLnJ)

Figure 72

# 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]](#_Chem_tablespace).

## Blood Bank

### TRX-01 Keep track of blood inventory

A table was created to satisfy this need [[LOCAL\_BLOOD\_BANK]](#_Chem_tablespace).

### 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 58. A NVL function was created to fill in the top row as “Grand Total” instead of (null) to avoid confusion.

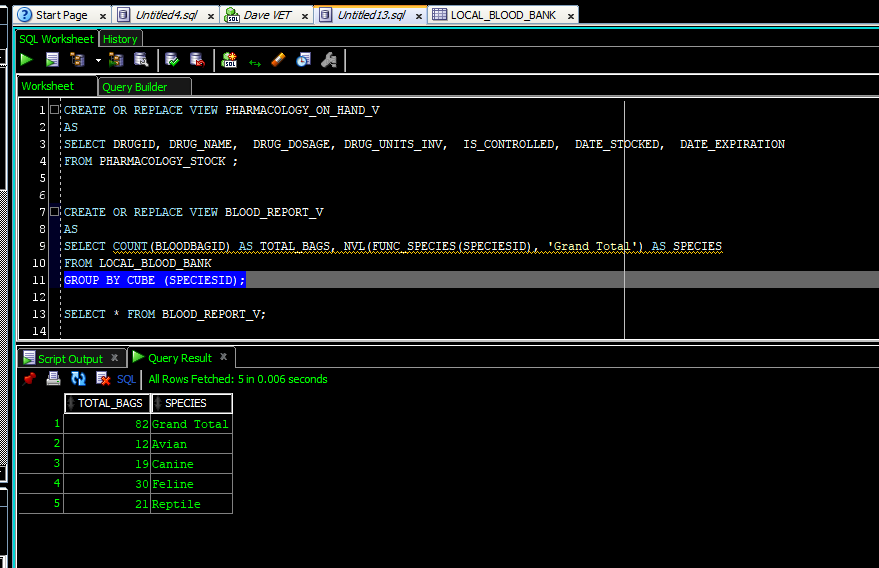
[](https://drive.google.com/open?id=1bJPWORK6yFHXqii6d15vbtWAuOE8WPGw)

Figure 73

## 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.
* INV-06: An invoice should be made from an estimate if the estimate is approved. This was not included after discussion with the clinic staff, there could be complications during clicnical procedures that may raise the price, and they would prefer to leave estimates as is.
* INV-07: Having an invoice number be the same as an approved estimate was not feasible at this time. Future edtions may address this issue.

# 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 59]. 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.

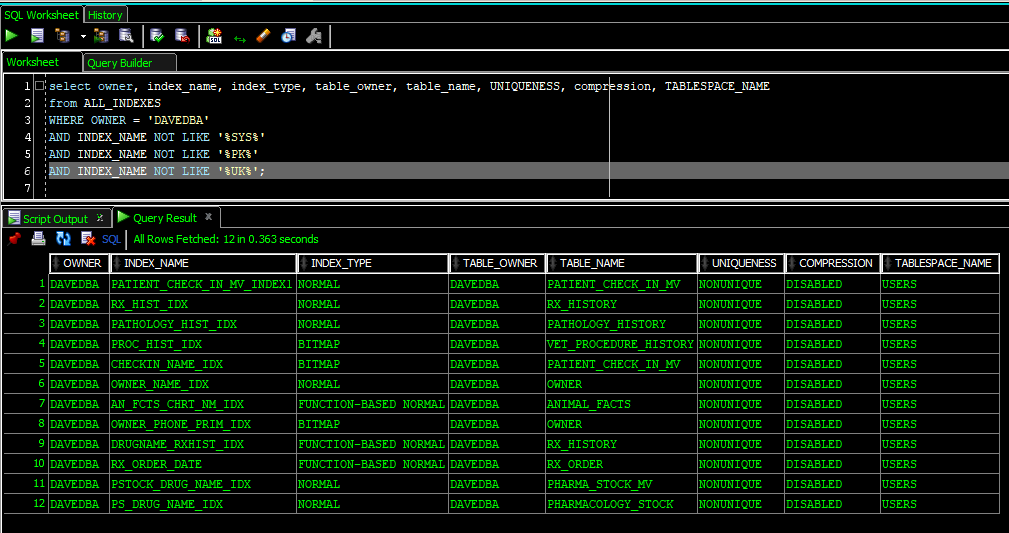
[](https://drive.google.com/open?id=1oi8eBx7CeV7NuVO0bhZwjmTDytgVb4Mj)

Figure 74

# DEPENDENCY ANALYSIS

Determining which tables and program units are dependent on other program units makes debugging the database and modifying structures much easier. The following images will show examples of dependency analysis from the DEPTREE\_FILL procedure included with Oracle’s database. **Error! Reference source not found.** shows a dependency analysis of the ANIMAL\_FACTS table, Figure 61 shows an analysis of the CHART\_PKG package, and Figure 62 shows the function FUNC\_PROCNAME and its dependencies.

[](https://drive.google.com/open?id=1FY6kz-Q7iZfIttFkc68gMUX9ubKfxDUl)

Figure 75

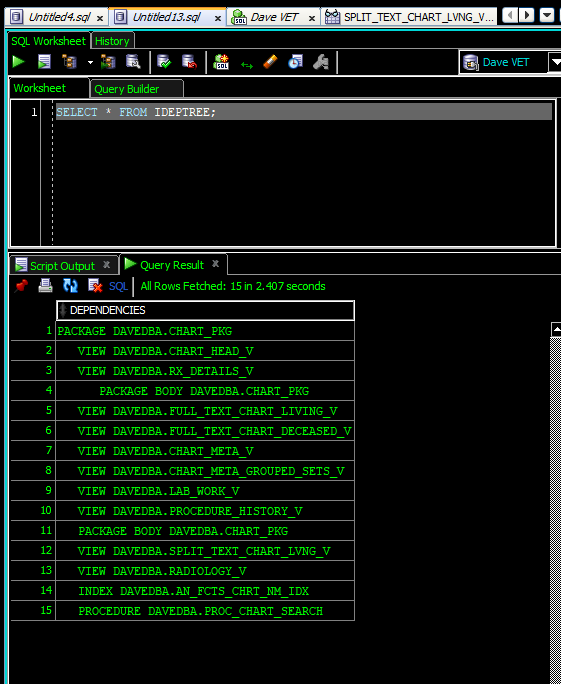
[](https://drive.google.com/open?id=1hE40WAXxtjF0rTEwqBNtbvZj34Cly2Ww)

Figure 76

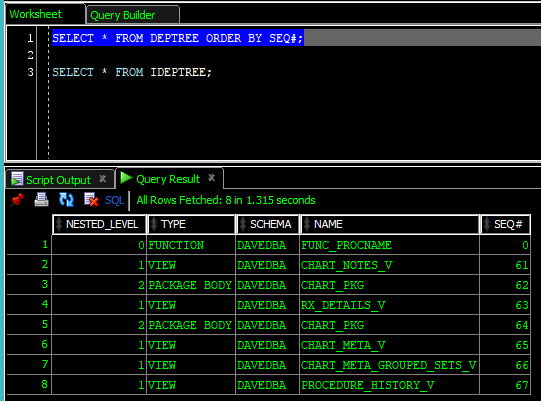
[](https://drive.google.com/open?id=1jrsLFMuTg50d3ftXQVO-IAG8fx3E_j2j)

Figure 77

# 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

### Tables

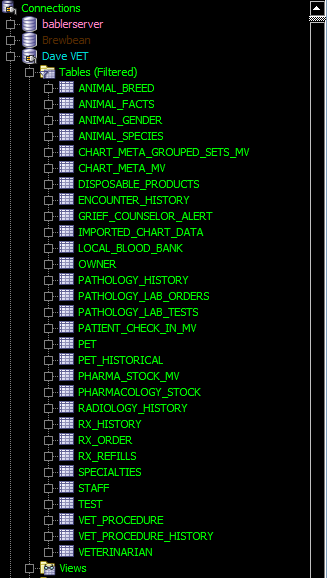
[](https://drive.google.com/open?id=18BUZqOGgEAjAwJgCJrgbwB4xeZSYwpQ8)

Figure 78

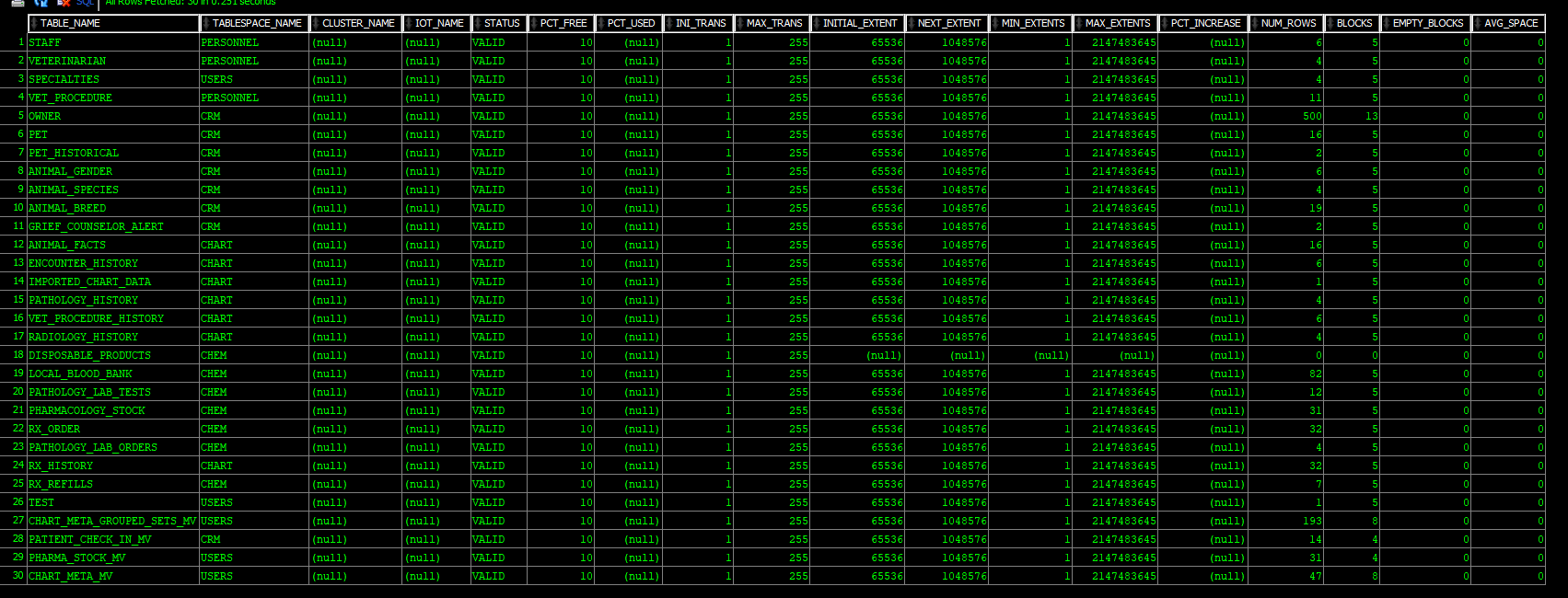
[](https://drive.google.com/open?id=1hWO5RwclI3slshx-MyP6JVYqmhL1SSPX)

Figure 79

### Materialized Views

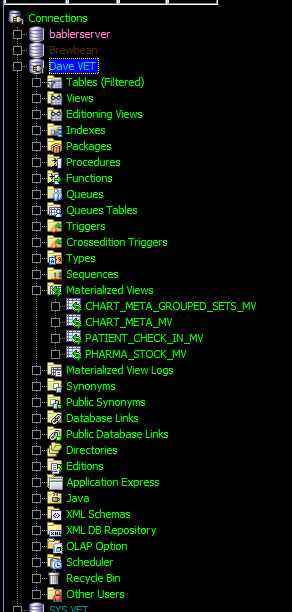
[](https://drive.google.com/open?id=1bXvmxgYO1lU2ih-gtIpAlj7ljcMl4Zxs)

Figure 80

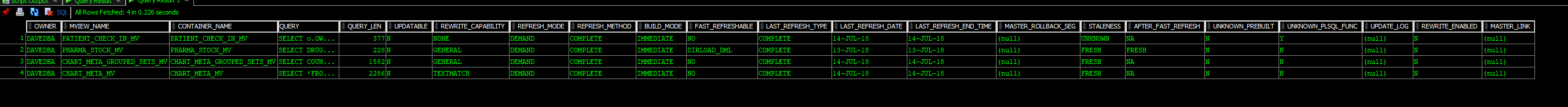
[](https://drive.google.com/open?id=1bXvmxgYO1lU2ih-gtIpAlj7ljcMl4Zxs)

Figure 81

### Views

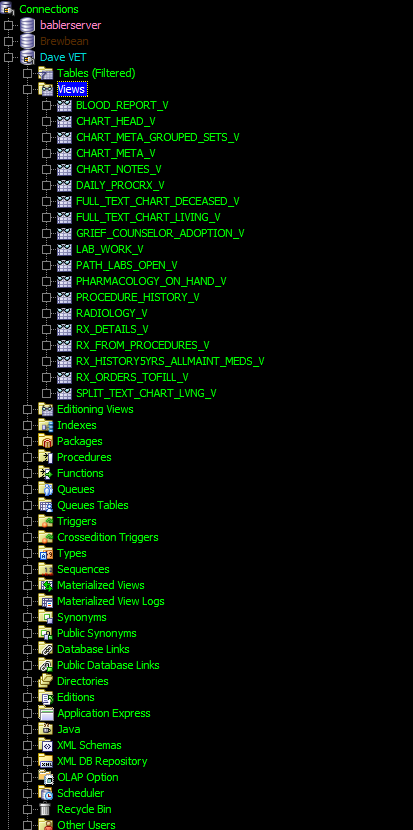
[](https://drive.google.com/open?id=1mCpAbsTNKsK0Xke4py7yhozML73s7V6V)

Figure 82

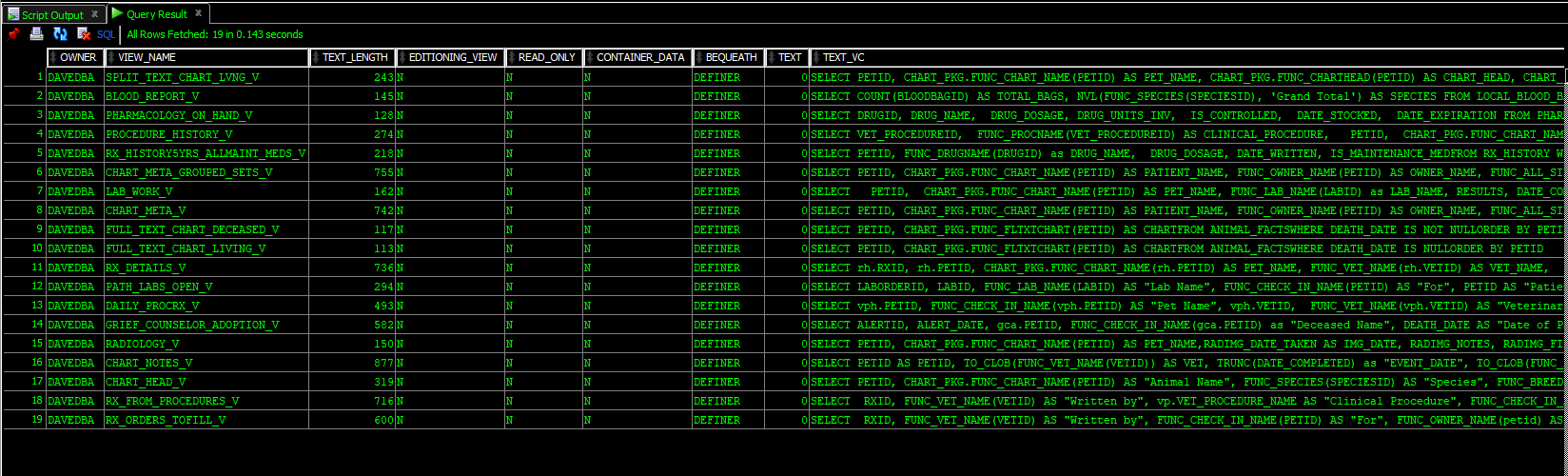
[](https://drive.google.com/open?id=1YrBBgdlEyaJLuaATR8VaoOeB3h2K01XL)

Figure 83

## Program Units

### Triggers

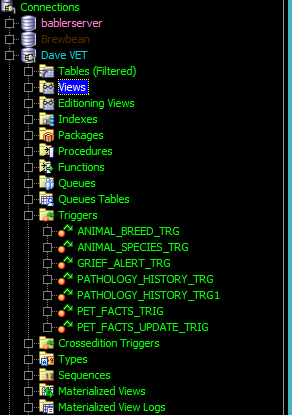
[](https://drive.google.com/open?id=1DMT6mWlhnJNsjlclB5XuFGZ54wnS8Wko)

Figure 84

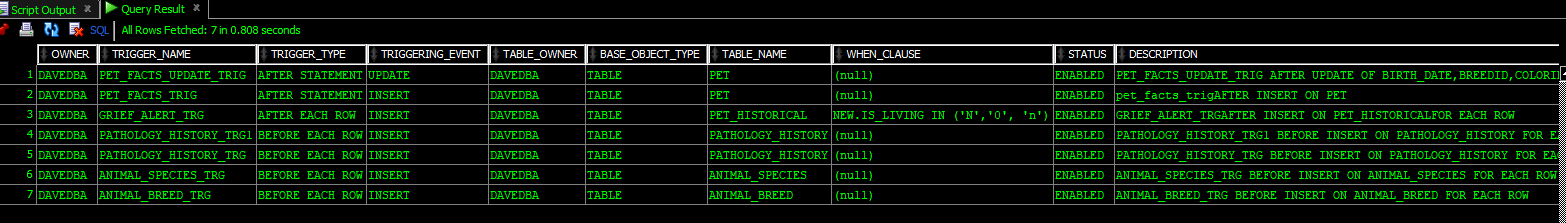
[](https://drive.google.com/open?id=1l_3hmvIdpTtcdbBb8xjNKjD1d5HN2KpK)

Figure 85

### Functions

[](https://drive.google.com/open?id=15GBDo0j-BS2TjBNlF3nfEAJ94Wz-f7bG)

Figure 86

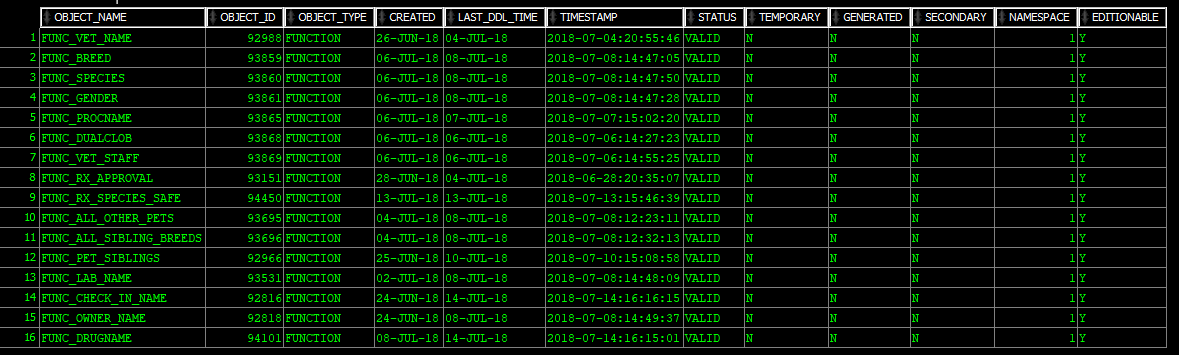
[](https://drive.google.com/open?id=1z9PJb2JrfTNB6eFwmiiUpA25LultgfTJ)

Figure 87

### Procedures

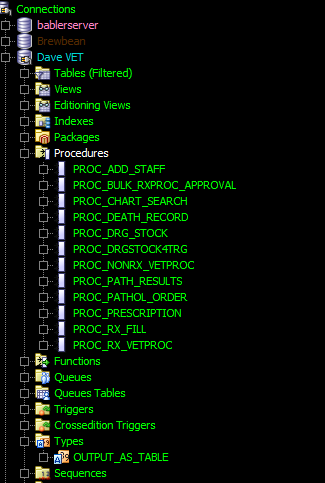
[](https://drive.google.com/open?id=1tQH1AfnFrhtr00oPbDA7tmWwLnJs_96w)

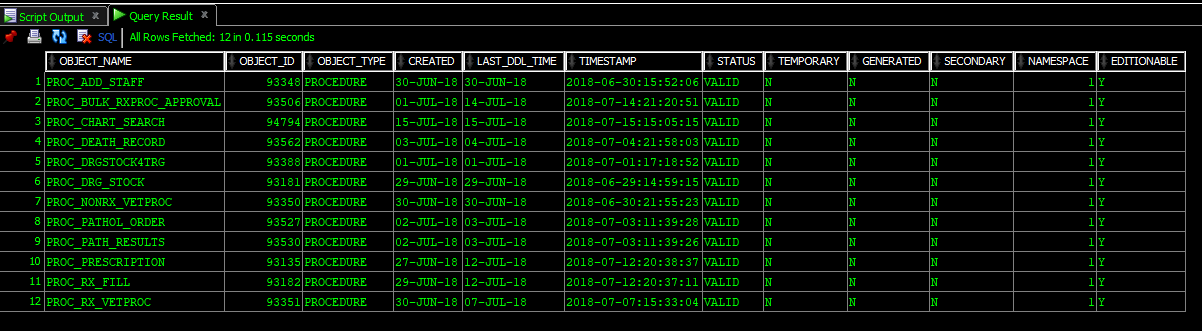
Figure 88[](https://drive.google.com/open?id=1GFec0DJP-6GppF4IHY3nvfWn3T2fb2kF)

Figure 89

### Packages

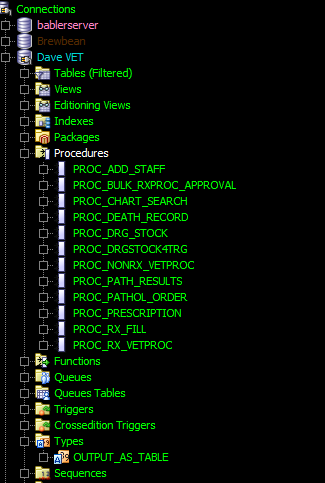
[](https://drive.google.com/open?id=16EoLQcnivNyJcyk1m-lzj1t91IyT0FVC)

Figure 90

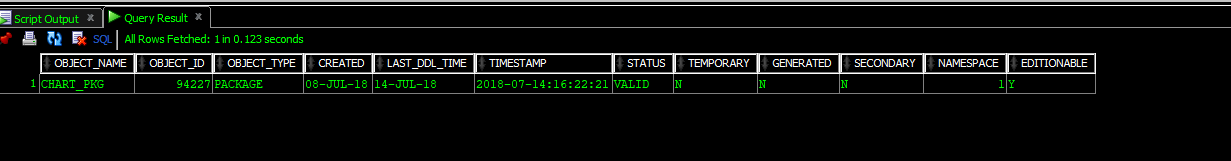
[](https://drive.google.com/open?id=1EiDT6W7AYLrPwLhUj6Dm5XyGC0yCb4v1)

Figure 91

## Sequences & Indices

### Sequences

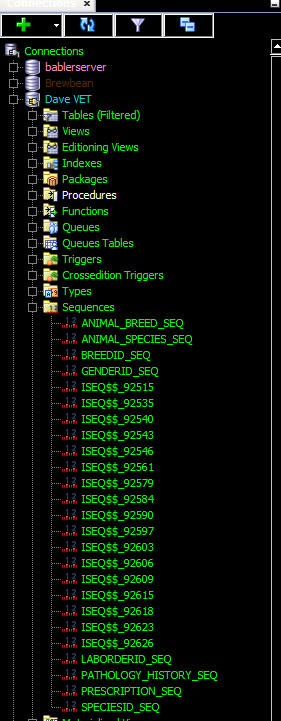
[](https://drive.google.com/open?id=1nviX8XEa2WE5zVCu6RK-iVcE0Z-nr16D)

Figure 92

[](https://drive.google.com/open?id=19dNIEyOsGEsaqUWrr-J3127Qoj-Jfh6P)

Figure 93

### Indices

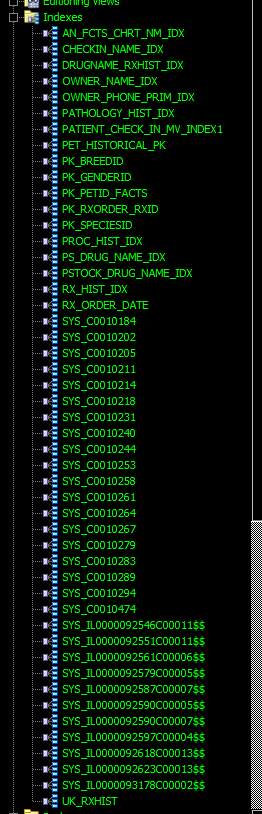
[](https://drive.google.com/open?id=1TKqVetqiRJxpWhZQWFlIn8_Ujp8VzQz7)

Figure 94

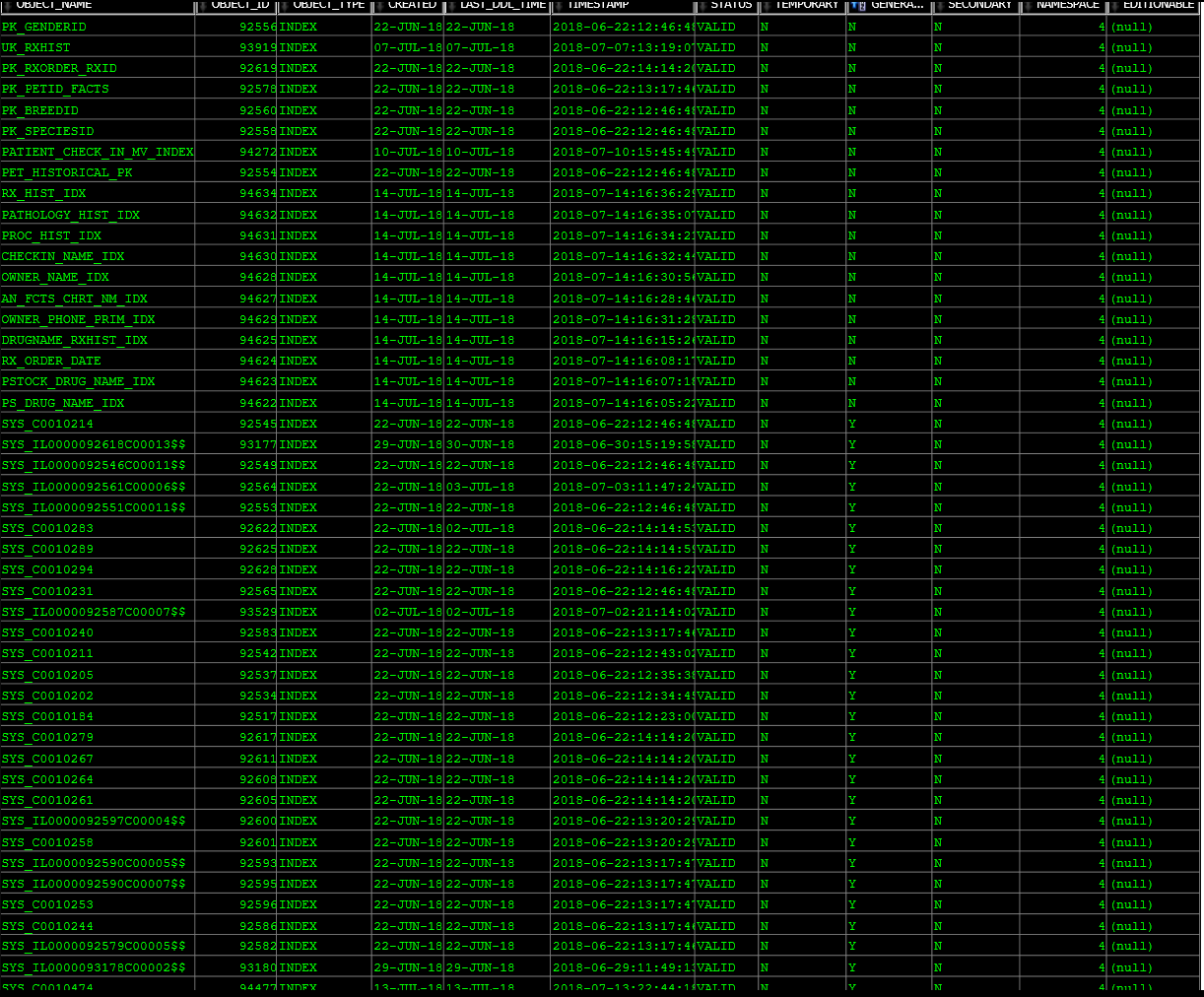
[](https://drive.google.com/open?id=1eg_vJ72QTrKB70lGv4PNMvk2HIZflxGk)

Figure 95

# 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

Code is copied from Microsoft’s Visual Studio: Code with high contrast settings for easy readability. The code is selectable and can be copied from the document.

### TABLES & DATA STRUCTURES

#### TABLES

##### Chart tablespace

--CHART

    --ANIMAL\_FACTS

**CREATE** **TABLE** ANIMAL\_FACTS(

    PetID   int,

    ChartID int, --if we don't purge we will create a trigger that makes it same as PetID

    Pet\_First\_Name  *varchar2*(40),

    Pet\_Middle\_Name *varchar2*(40),

    Owner\_Last\_Name *varchar2*(40),

    SpeciesID int,

    BreedID int,

    GenderID int,

    Coloring *varchar2*(30),

    Birth\_Date  *date*,

    Temperament\_Notes *varchar2*(80),

    Chart\_Create\_Date *date*,

**CONSTRAINT** pk\_petID\_facts **PRIMARY KEY** (PetID),

**CONSTRAINT** fk\_petID\_facts **FOREIGN KEY** (PetID)

**REFERENCES** PET(PetID),

**CONSTRAINT** fk\_speciesID\_facts **FOREIGN KEY** (SpeciesID)

**REFERENCES** ANIMAL\_SPECIES(SpeciesID),

**CONSTRAINT** fk\_breedid\_facts **FOREIGN KEY** (BreedID)

**REFERENCES** ANIMAL\_BREED(BreedID),

**CONSTRAINT** fk\_genderid\_facts **FOREIGN KEY** (GenderID)

**REFERENCES** ANIMAL\_GENDER(GenderID))

**TABLESPACE** CHART;

**COMMENT** **ON** **TABLE** ANIMAL\_FACTS **IS** '1 of 2 ANIMAL\_FACTS is a CHILD of PET from the CRM tablespace. 2 of 2 ANIMAL\_FACTS is the foundation for the patient chart';

**COMMENT** **ON** **COLUMN** ANIMAL\_FACTS.PetID **IS** 'ALL chart objects reference the PetID in the CHART tablespace (so this one) for simplicity, and to link them together logically';

    --ENCOUNTER\_HISTORY

**CREATE** **TABLE** ENCOUNTER\_HISTORY(

        EncounterID int GENERATED **AS** IDENTITY **PRIMARY KEY** **NOT** NULL,

        PetID   int,

        Encounter\_Weight *number*(8,2),

        VetID   int,

        Encounter\_Notes *clob*,

        Encounter\_Date\_Time *timestamp*(5),

**CONSTRAINT** fk\_petid\_encounter **FOREIGN KEY** (PetID)

**REFERENCES** ANIMAL\_FACTS(PetID),

**CONSTRAINT** fk\_vetid\_encounter **FOREIGN KEY** (VetID)

**REFERENCES** VETERINARIAN(VetID))

**TABLESPACE** CHART;

    --IMPORTED\_CHART\_DATA

**CREATE** **TABLE** IMPORTED\_CHART\_DATA(

        PetID int,

        ImportID int GENERATED **AS** IDENTITY **PRIMARY KEY** **NOT** NULL,

        Import\_Files *bfile*,

**CONSTRAINT** fk\_petid\_import **FOREIGN KEY** (PetID)

**REFERENCES** ANIMAL\_FACTS(PetID))

**TABLESPACE** CHART;

    --RADIOLOGY\_HISTORY

**CREATE** **TABLE** RADIOLOGY\_HISTORY(

        PetID   int,

        RadImgID int GENERATED **AS** IDENTITY **PRIMARY KEY** **NOT** NULL,

        RadImg\_Date\_Taken   *date*,

        RadImg\_Notes    *clob*,

        RadImg\_Files    *bfile*,

**CONSTRAINT** fk\_petid\_rad **FOREIGN KEY** (PetID)

**REFERENCES** ANIMAL\_FACTS(PetID))

**TABLESPACE** CHART;

    --PATHOLOGY\_HISTORY

**CREATE** **TABLE** PATHOLOGY\_HISTORY(

        LabHistoryID int GENERATED **AS** IDENTITY **PRIMARY KEY** **NOT** NULL,

        LabOrderID int,

        PetID int,

        LabID int,

        VetID int,

        Critical\_Disease *char*(1),

        Date\_Completed *date*,

        Results *varchar2*(1000),

**CONSTRAINT** fk\_petid\_labhist **FOREIGN KEY** (PetID)

**REFERENCES** ANIMAL\_FACTS(PetID),

**CONSTRAINT** fk\_vetid\_labhist **FOREIGN KEY** (VetID)

**REFERENCES** VETERINARIAN(VetID),

**CONSTRAINT** ck\_crit\_disease **CHECK** (Critical\_Disease **IN** ('Y', 'y', 'N', 'n', 0, 1)))

**TABLESPACE** CHART;

    --VET\_PROCEDURE\_HISTORY

**CREATE** **TABLE** VET\_PROCEDURE\_HISTORY(

        Patient\_Vet\_ProcedureID int GENERATED **AS** IDENTITY **PRIMARY KEY** **NOT** NULL,

        Vet\_ProcedureID int,

        PetID int,

        Vet\_Procedure\_Date *date*,

        Vet\_Procedure\_Notes *clob*,

        Vet\_Procedure\_FollowUp\_Date *date*,

        Vet\_Procedure\_FollowUp\_Outcome *clob*,

        Administered\_Rx\_During *char*(1),

        VetID int,

**CONSTRAINT** fk\_vprocid\_hist **FOREIGN KEY** (Vet\_ProcedureID)

**REFERENCES** VET\_PROCEDURE(Vet\_ProcedureID),

**CONSTRAINT** fk\_petid\_prochist **FOREIGN KEY** (PetID)

**REFERENCES** ANIMAL\_FACTS(PetID),

**CONSTRAINT** fk\_vetid\_prochist **FOREIGN KEY** (VetID)

**REFERENCES** VETERINARIAN(VetID),

**CONSTRAINT** ck\_adminrx\_proc **CHECK** (Administered\_Rx\_During **IN** ('Y', 'y', 'N', 'n', 0, 1)))

**TABLESPACE** CHART;

    --RX\_HISTORY

    --can't add Patient\_Vet\_ProcedureID FOREIGN KEY until after Vet\_Procedure\_History Has been loaded

**CREATE** **TABLE** RX\_HISTORY(

        RxID int GENERATED **AS** IDENTITY **PRIMARY KEY** **NOT** NULL,

        PetID int,

        DrugID int,

        Drug\_Dosage *number*(9,2),

        Drug\_Units\_Dispensed *number*(9,2),

        Date\_Filled *date*,

        Patient\_Vet\_ProcedureID int,

        Is\_Maintenance\_Med *char*(1),

        Notes *varchar2*(1000),

**CONSTRAINT** fk\_petid\_rxhist **FOREIGN KEY** (PetID)

**REFERENCES** ANIMAL\_FACTS(PetID),

**CONSTRAINT** fk\_drugid\_rxhist **FOREIGN KEY** (DrugID)

**REFERENCES** PHARMACOLOGY\_STOCK(DrugID),

**CONSTRAINT** fk\_pvproc\_rxhist **FOREIGN KEY** (Patient\_Vet\_ProcedureID)

**REFERENCES** VET\_PROCEDURE\_HISTORY(Patient\_Vet\_ProcedureID),

**CONSTRAINT** ck\_ismaintmed **CHECK** **IN** (Is\_Maintenance\_Med **IN** ('Y', 'y', 'N', 'n', 0, 1)))

**TABLESPACE** CHART;

##### Chem tablespace

--CHEMICAL/PHARMA

    --DISPOSABLE\_PRODUCTS

**CREATE** **TABLE** DISPOSABLE\_PRODUCTS(

            ProductID int GENERATED **AS** IDENTITY **PRIMARY KEY** **NOT** NULL,

            Product\_Description *varchar2*(40),

            Product\_Size *varchar2*(10),

            Product\_On\_Hand int)

**TABLESPACE** CHEM;

    --LOCAL\_BLOOD\_BANK

**CREATE** **TABLE** LOCAL\_BLOOD\_BANK(

            BloodBagID int GENERATED **AS** IDENTITY **PRIMARY KEY** **NOT** NULL,

            SpeciesID int,

            Type\_Blood *varchar2*(40),

**CONSTRAINT** fk\_SpeciesID **FOREIGN KEY** (SpeciesID)

**REFERENCES** ANIMAL\_SPECIES(SpeciesID),

**CONSTRAINT** ck\_Type\_Blood **CHECK** (Type\_Blood **IN** ('A', 'B', 'AB', '-', '+', 'universal')))

**TABLESPACE** CHEM;

--LAB SECTION

   --PATHOLOGY\_LAB\_TESTS

**CREATE** **TABLE** PATHOLOGY\_LAB\_TESTS(

        LabID int GENERATED **AS** IDENTITY **PRIMARY KEY** **NOT** NULL,

        Lab\_Name *varchar2*(40),

        Lab\_Cost *number*(7,2),

        Kits\_on\_Hand int)

**TABLESPACE** CHEM;

    --PATHOLOGY\_LAB\_ORDERS

**CREATE** **TABLE** PATHOLOGY\_LAB\_ORDERS(

            LabOrderID int GENERATED **AS** IDENTITY **PRIMARY KEY** **NOT** NULL,

            LabID int,

            PetID int,

            VetID int,

            Date\_Completed *date*,

**CONSTRAINT** fk\_labid **FOREIGN KEY** (LabID)

**REFERENCES** PATHOLOGY\_LAB\_TESTS(LabID),

**CONSTRAINT** fk\_petid\_lab **FOREIGN KEY** (PetID)

**REFERENCES** ANIMAL\_FACTS(PetID),

**CONSTRAINT** fk\_vetID\_laborders **FOREIGN KEY** (VetID)

**REFERENCES** VETERINARIAN(VetID))

**TABLESPACE** CHEM;

        /\*WARNING, YOU CANNOT LOAD THIS TABLE UNTIL SOME OF THE CHART HAS BEN MADE\*/

    --PHARMACOLOGY\_STOCK

**CREATE** **TABLE** PHARMACOLOGY\_STOCK(

            DrugID  int GENERATED **AS** IDENTITY **PRIMARY KEY** **NOT** NULL,

            Drug\_Name *varchar2*(60),

            Drug\_Dosage *number*(9,2),

            Drug\_Units\_Inv *number*(9,2),

            Drug\_Units\_Meas *varchar2*(20),

            Drug\_Cost\_Per\_Unit  *number*(7,2),

            Is\_Controlled   *char*(1),

            Avian\_Safe  *char*(1),

            Canine\_Safe *char*(1),

            Feline\_Safe *char*(1),

            Reptile\_Safe *char*(1),

            Date\_Stocked *date*,

            Date\_Expiration *date*,

            Order\_Level *number*(7,2),

            Reorder\_Flag *char*(1),

**CONSTRAINT** ck\_is\_controlled **CHECK** (Is\_Controlled **IN** ('Y', 'y', 'N', 'n', 0, 1)),

**CONSTRAINT** ck\_aviansafe **CHECK** (Avian\_Safe **IN** ('Y', 'y', 'N', 'n', 0, 1)),

**CONSTRAINT** ck\_caninesafe **CHECK** (Canine\_Safe **IN** ('Y', 'y', 'N', 'n', 0, 1)),

**CONSTRAINT** ck\_felinesafe **CHECK** (Feline\_Safe **IN** ('Y', 'y', 'N', 'n', 0, 1)),

**CONSTRAINT** ck\_reptilesafe **CHECK** (Reptile\_Safe **IN** ('Y', 'y', 'N', 'n', 0, 1)),

**CONSTRAINT** ck\_reorder **CHECK** (Reorder\_Flag **IN** ('Y', 'y', 'N', 'n', 0, 1)))

**TABLESPACE** CHEM;

    --RX\_ORDER

    /\*note: do not create FORIEGN KEYS for

    RxID

    until Tablespace Chart's Rx\_History is created\*/

**CREATE** **TABLE** RX\_ORDER(

            RxID int,

            VetID int,

            PetID int,

            Date\_Submitted *date*,

            DrugID int,

            Drug\_Units\_Prescribed *number*(9,2),

            Drug\_Units\_Dispensed *number*(9,2),

            Vet\_Procedure\_ID int,

            Date\_Filled *date*,

**CONSTRAINT** pk\_rxorder\_rxid **PRIMARY KEY** (RxID),

**CONSTRAINT** fk\_drug\_rxorder **FOREIGN KEY** (DrugID)

**REFERENCES** PHARMACOLOGY\_STOCK(DrugID))

**TABLESPACE** CHEM;

**COMMENT** **ON** **TABLE** RX\_ORDER **IS** 'RX\_ORDER is a child table of RX\_HISTORY';

    --RX\_REFILLS

        ---The constraints for this table can load immediately after RX\_ORDER even if it's constraints are not ready

**CREATE** **TABLE** RX\_REFILLS (

            RefillID int GENERATED **AS** IDENTITY **PRIMARY KEY** **NOT** NULL,

            RxOrderID int,

            RxID int,

            Num\_Refills\_Left int,

            Date\_Filled *date*,

**CONSTRAINT** pk\_rx\_refillid **PRIMARY KEY** (RefillID))

**TABLESPACE** CHEM;

--add after installing RX\_history table

**ALTER** **TABLE** RX\_REFILLS --make sure that RX\_History is in first.

**ADD** **CONSTRAINT** fk\_rxid\_refills **FOREIGN KEY** (RxID)

**REFERENCES** RX\_ORDER(RxID);

**ALTER** **TABLE** RX\_ORDER

**ADD** **CONSTRAINT** fk\_rxidhist **FOREIGN KEY** (RxID)

**REFERENCES** RX\_HISTORY(RxID);

##### CRM tables

**DROP** **TABLE** OWNER CASCADE CONSTRAINTS;

**DROP** **TABLE** PET CASCADE CONSTRAINTS;

**DROP** **TABLE** PET\_HISTORICAL CASCADE CONSTRAINTS;

**DROP** **TABLE** ANIMAL\_GENDER CASCADE CONSTRAINTS;

**DROP** **TABLE** ANIMAL\_SPECIES CASCADE CONSTRAINTS;

**DROP** **TABLE** ANIMAL\_BREED CASCADE CONSTRAINTS;

**DROP** **TABLE** GRIEF\_COUNSELOR\_ALERT CASCADE CONSTRAINTS;

    --OWNER

**CREATE** **TABLE** OWNER(

        OwnerID int GENERATED **AS** IDENTITY **PRIMARY KEY** **NOT** NULL,

        First\_Name *varchar2*(40),

        Last\_Name *varchar2*(40),

        Phone\_Primary *varchar2*(9),

        Phone\_Secondary *varchar2*(9),

        Address\_Street *varchar2*(60),

        Address\_Apt *varchar2*(10),

        City *varchar2*(40),

        State *char*(2),

        Zip *char*(2),

        Email *varchar2*(50),

        Alt\_Family\_Mem\_First\_Name *varchar2*(40),

        Alt\_Family\_Mem\_Last\_Name *varchar2*(40),

        Alt\_Family\_Mem\_Phone *varchar2*(9),

        Emerg\_Cont\_First\_Name *varchar2*(40),

        Emerg\_Cont\_Last\_Name *varchar2*(40),

        Emerg\_Cont\_Phone *varchar2*(9))

**TABLESPACE** CRM;

    --PET

**CREATE** **TABLE** PET(

        PetID int GENERATED **AS** IDENTITY **PRIMARY KEY** **NOT** NULL,

        OwnerID int **NOT** NULL,

        Pet\_First\_Name *varchar2*(40),

        Pet\_Middle\_Name *varchar2*(40),

        /\*The next 3 lines of "IDs" are Foreign Keys that cannot yet be created,

        will need to create after the appropriate tables are created\*/

        SpeciesID int,

        BreedID int,

        GenderID int,

        Coloring *varchar2*(30),

        Birth\_Date *date*,

        Is\_Living *char*(1),

        Photo *blob*,

        Temperament\_Notes *varchar2*(80),

**CONSTRAINT** ck\_is\_living **CHECK** (is\_living **IN** ('Y', 'y', 'N', 'n', 0, 1))

        )

**TABLESPACE** CRM;

**COMMENT** **ON** **COLUMN** PET.Is\_Living **IS** 'Oracle does not support BOOLEAN attributes, this is a pseudoBoolean to create a flag, currently unknown if the programmers will use upper or lower case, or a 1|0 to set the flag have taken all into account';

    --PET\_HISTORICAL

**CREATE** **TABLE** PET\_HISTORICAL(

        PetID int **NOT** NULL,

        OwnerID int,

        Pet\_First\_Name *varchar2*(40),

        Pet\_Middle\_Name *varchar2*(40),

        /\*The next 3 lines of "IDs" are Foreign Keys that cannot yet be created,

        will need to create after the appropriate tables are created\*/

        SpeciesID int,

        BreedID int,

        GenderID int,

        Coloring *varchar2*(30),

        Birth\_Date *date*,

        Is\_Living *char*(1),

        Photo *blob*,

        Temperament\_Notes *varchar2*(80),

**CONSTRAINT** pet\_historical\_pk **PRIMARY KEY** (PetID),

**CONSTRAINT** check\_is\_living\_historical **CHECK** (is\_living **IN** ('N', 'n', 0))

        )

**TABLESPACE** CRM;

**COMMENT** **ON** **TABLE** PET\_HISTORICAL **IS** 'This is where archived data about dead pets is stored';

    --ANIMAL\_GENDER

**CREATE** **TABLE** ANIMAL\_GENDER(

        GenderID int **NOT** NULL,

        Gender\_Name *varchar2*(25),

**CONSTRAINT** pk\_genderID **PRIMARY KEY** (GenderID))

**TABLESPACE** CRM;

    --ANIMAL\_SPECIES

**CREATE** **TABLE** ANIMAL\_SPECIES(

        SpeciesID int **NOT** NULL,

        Species\_Name *varchar2*(25),

**CONSTRAINT** pk\_speciesID **PRIMARY KEY** (SpeciesID))

**TABLESPACE** CRM;

    --ANIMAL\_BREED

**CREATE** **TABLE** ANIMAL\_BREED(

            BreedID int **NOT** NULL,

            SpeciesID int **NOT** NULL,

            Breed\_Name *varchar2*(25),

**CONSTRAINT** pk\_breedID **PRIMARY KEY** (BreedID),

**CONSTRAINT** fk\_species\_breed **FOREIGN KEY** (SpeciesID)

**REFERENCES** ANIMAL\_SPECIES(SpeciesID))

**TABLESPACE** CRM;

    --GRIEF\_COUNSELOR\_ALERT

**CREATE** **TABLE** GRIEF\_COUNSELOR\_ALERT(

        AlertID int GENERATED **AS** IDENTITY **PRIMARY KEY**,

        Alert\_Date *date*,

        PetID int,

        OwnerID int,

        Parent\_Last *varchar2*(40),

        Pet\_First *varchar2*(40),

        Complete\_Date *date*,

        Resolution\_Notes *clob*,

        Phone\_Primary *varchar2*(9),

        Death\_Date *date*,

**CONSTRAINT** fk\_pet\_grief **FOREIGN KEY** (PetID)

**REFERENCES** PET\_HISTORICAL(PetID),

**CONSTRAINT** fk\_owner\_grief **FOREIGN KEY** (OwnerID)

**REFERENCES** OWNER(OwnerID))

**TABLESPACE** CRM;

**COMMENT** **ON** **COLUMN** GRIEF\_COUNSELOR\_ALERT.PetID **IS** 'the pet has died don''t use the living pet table for a reference';

###### Estimates & Invoicing (part of crm)

**CREATE** **TABLE** ESTIMATE(

LineID int GENERATED ALWAYS **AS** IDENTITY,

EstimateID int,

PetID int,

VetID int,

Date\_Estimate\_Creation *date*,

DrugID int,

Drug\_Cost\_Per\_Unit *number*(7,2),

Drug\_Units\_Prescribed int,

Rx\_Cost *number*(7,2),

LabID int,

Lab\_Cost *number*(7,2),

Specialty\_Add\_On\_Cost *number*(7,2),

SpecialtyID int,

Vet\_ProcedureID int,

OR\_Fee *number* (5,2),

Procedure\_Cost *number*(7,2),

Add\_Ons\_Description *varchar2*(2000),

Invididual\_Add\_On\_Cost *number*(7,2),

Total\_Add\_On\_Costs *number*(7,2),

Total\_Estimate\_Cost *number*(9,2),

Late\_Fee *number*(7,2),

**CONSTRAINT** fk\_petid\_esto **FOREIGN KEY** (PETID)

**REFERENCES** ANIMAL\_FACTS(PETID),

**CONSTRAINT** fk\_drugid\_esto **FOREIGN KEY** (DRUGID)

**REFERENCES** PHARMACOLOGY\_STOCK(DRUGID),

**CONSTRAINT** fk\_labid\_esto **FOREIGN KEY** (LABID)

**REFERENCES** PATHOLOGY\_LAB\_TESTS(LABID),

**CONSTRAINT** fk\_specialtyid\_esto **FOREIGN KEY** (SPECIALTYID)

**REFERENCES** SPECIALTIES(SPECIALTYID),

**CONSTRAINT** fk\_vetprocedureid\_esto **FOREIGN KEY** (VET\_PROCEDUREID)

**REFERENCES** VET\_PROCEDURE(VET\_PROCEDUREID)

**CONSTRAINT** ck\_oneline\_esto **CHECK**

        ((DRUGID IS NOT NULL **AND** (VET\_PROCEDUREID IS NULL **AND** LABID IS NULL **AND** Add\_Ons\_Description IS NULL))

**OR**

            (LABID IS NOT NULL **AND** (DRUGID IS NULL **AND** VET\_PROCEDUREID IS NULL **AND** Add\_Ons\_Description IS NULL))

**OR**

            (VET\_PROCEDUREID IS NOT NULL **AND** (DRUGID IS NULL **AND** LABID IS NULL **AND** Add\_Ons\_Description IS NULL))

**OR**

            (Add\_Ons\_Description IS NOT NULL **AND** (DRUGID IS NULL **AND** LABID IS NULL **AND** VET\_PROCEDUREID IS NULL))

**OR** DRUGID IS NULL **AND** VET\_PROCEDUREID IS NULL **AND** LABID IS NULL **AND** Add\_Ons\_Description IS NULL)

)

**TABLESPACE** CRM;

**COMMENT** **ON** **COLUMN** ESTIMATE.EstimateID **IS** 'Generated by estimate\_seq, through a procedure, is how grouping will work for reports';

()

**CREATE** **TABLE** INVOICE\_OPEN(

LineID int GENERATED **AS** IDENTITY **PRIMARY KEY** **NOT** NULL,

InvoiceID int,

PetID int,

VetID int,

Date\_Invoice\_Creation *date*,

RXID int,

DrugID int,

Drug\_Cost\_Per\_Unit *number*(7,2),

Drug\_Units\_Prescribed int,

Rx\_Cost *number*(7,2),

LabID int,

Lab\_Cost *number*(7,2),

Specialty\_Add\_On\_Cost *number*(7,2),

SpecialtyID int,

Vet\_ProcedureID int,

OR\_Fee *number* (5,2),

Procedure\_Cost *number*(7,2),

Add\_Ons\_Description *varchar2*(2000),

Invididual\_Add\_On\_Cost *number*(7,2),

LINE\_SUBTOTAL *number*(9,2),

GRAND\_TOTAL *number*(9,2),

**CONSTRAINT** fk\_rxid\_inv\_o **FOREIGN KEY** (RXID)

**REFERENCES** RX\_HISTORY(RXID),

**CONSTRAINT** fk\_petid\_inv\_o **FOREIGN KEY** (PETID)

**REFERENCES** ANIMAL\_FACTS(PETID),

**CONSTRAINT** fk\_drugid\_inv\_o **FOREIGN KEY** (DRUGID)

**REFERENCES** PHARMACOLOGY\_STOCK(DRUGID),

**CONSTRAINT** fk\_labid\_inv\_o **FOREIGN KEY** (LABID)

**REFERENCES** PATHOLOGY\_LAB\_TESTS(LABID),

**CONSTRAINT** fk\_specialtyid\_inv\_o **FOREIGN KEY** (SPECIALTYID)

**REFERENCES** SPECIALTIES(SPECIALTYID),

**CONSTRAINT** fk\_vetprocedureid\_inv\_o **FOREIGN KEY** (VET\_PROCEDUREID)

**REFERENCES** VET\_PROCEDURE(VET\_PROCEDUREID),

**CONSTRAINT** ck\_ispaid **CHECK** (Paid\_In\_Full **IN** ('Y', 'y', 'T', 't', 'F', 'f', 'N', 'n', '0', '1')),

**CONSTRAINT** ck\_oneline\_inv\_o **CHECK**

            ((DRUGID IS NOT NULL **AND** (VET\_PROCEDUREID IS NULL **AND** LABID IS NULL **AND** Add\_Ons\_Description IS NULL))

**OR**

            (LABID IS NOT NULL **AND** (DRUGID IS NULL **AND** VET\_PROCEDUREID IS NULL **AND** Add\_Ons\_Description IS NULL))

**OR**

            (VET\_PROCEDUREID IS NOT NULL **AND** (DRUGID IS NULL **AND** LABID IS NULL **AND** Add\_Ons\_Description IS NULL))

**OR**

            (Add\_Ons\_Description IS NOT NULL **AND** (DRUGID IS NULL **AND** LABID IS NULL **AND** VET\_PROCEDUREID IS NULL))

**OR** DRUGID IS NULL **AND** VET\_PROCEDUREID IS NULL **AND** LABID IS NULL **AND** Add\_Ons\_Description IS NULL)

)

)

**TABLESPACE** CRM;

**COMMENT** **ON** **COLUMN** INVOICE\_OPEN.InvoiceID **IS** 'Generated by invoice\_seq, through a procedure, is how grouping will work for reports';

**CREATE** **TABLE** PAYMENT\_TYPE(

Payment\_TypeID int GENERATED **AS** IDENTITY **PRIMARY KEY** **NOT** NULL,

Payment\_Type\_Name *varchar2*(64)

    )

**TABLESPACE** CRM;

**CREATE** **TABLE** PAYMENT\_VERIFICATION(

PaymentID int GENERATED **AS** IDENTITY **PRIMARY KEY** **NOT** NULL,

InvoiceID int,

Payment\_TypeID int,

Amount\_Paid *number*(9,2),

Check\_Number int,

CC\_Encrypted12 *RAW*(240),

CC\_Last4 *varchar2*(4),

DATE\_PAID *date* **DEFAULT** SYSDATE,

/\*CONSTRAINT fk\_payver\_invid FOREIGN KEY (InvoiceID)

        REFERENCES INVOICE\_OPEN(InvoiceID), not actually a PK, may

        ultimately set up a connection table but for now PLSQL will suffice \*/

**CONSTRAINT** fk\_paymenttypeid **FOREIGN KEY** (Payment\_TypeID)

**REFERENCES** PAYMENT\_TYPE(Payment\_TypeID)

    )

**TABLESPACE** CRM;

**COMMENT** **ON** **COLUMN** PAYMENT\_VERIFICATION.CC\_Encrypted12 **IS** 'This holds the raw encrypted part of the credit card';

**CREATE** **TABLE** INVOICE\_CLOSED(

LineID int,

InvoiceID int,

PetID int,

VetID int,

Date\_Invoice\_Creation *date*,

RXID int,

DrugID int,

Drug\_Cost\_Per\_Unit *number*(7,2),

Drug\_Units\_Prescribed int,

Rx\_Cost *number*(7,2),

LabID int,

Lab\_Cost *number*(7,2),

Specialty\_Add\_On\_Cost *number*(7,2),

SpecialtyID int,

Vet\_ProcedureID int,

OR\_Fee *number* (5,2),

Procedure\_Cost *number*(7,2),

Add\_Ons\_Description *varchar2*(2000),

Invididual\_Add\_On\_Cost *number*(7,2),

LINE\_SUBTOTAL *number*(9,2),

Late\_Fee *number*(9,2),

GRAND\_TOTAL *number*(9,2),

**CONSTRAINT** fk\_rxid\_inv\_c **FOREIGN KEY** (RXID)

**REFERENCES** RX\_HISTORY(RXID),

**CONSTRAINT** fk\_petid\_inv\_c **FOREIGN KEY** (PETID)

**REFERENCES** ANIMAL\_FACTS(PETID),

**CONSTRAINT** fk\_drugid\_inv\_c **FOREIGN KEY** (DRUGID)

**REFERENCES** PHARMACOLOGY\_STOCK(DRUGID),

**CONSTRAINT** fk\_labid\_inv\_c **FOREIGN KEY** (LABID)

**REFERENCES** PATHOLOGY\_LAB\_TESTS(LABID),

**CONSTRAINT** fk\_specialtyid\_inv\_c **FOREIGN KEY** (SPECIALTYID)

**REFERENCES** SPECIALTIES(SPECIALTYID))

**TABLESPACE** CRM;

**COMMENT** **ON** **COLUMN** INVOICE\_CLOSED.LineID **IS** 'This data was generated as a PRIMARY KEY by INVOICE OPEN, and is superflous but kept for now';

##### Personnel tables

--STAFFING

    --STAFF

**CREATE** **TABLE** STAFF(

        StaffID int GENERATED **AS** IDENTITY **PRIMARY KEY**,

        Staff\_First\_Name *varchar2*(40),

        Staff\_Last\_Name *varchar2*(40),

        Employment\_Date *date*,

        Termination\_Date *date*,

        Is\_Rehireable *char*(1),

        Is\_Vet *char*(1),

        Database\_Role *varchar2*(40),

**CONSTRAINT** chk\_rehire **CHECK** (Is\_Rehireable **IN** ('Y', 'y', 'N', 'n', 0, 1)),

**CONSTRAINT** chk\_vet **CHECK** (Is\_Vet **IN** ('Y', 'y', 'N', 'n', 0, 1)))

**TABLESPACE** PERSONNEL;

**COMMENT** **ON** **COLUMN** STAFF.Is\_Rehireable **IS** 'Oracle does not support BOOLEAN attributes, this is a pseudoBoolean to create a flag, currently unknown if the programmers will use upper or lower case, or a 1|0 to set the flag have taken all into account';

**COMMENT** **ON** **COLUMN** STAFF.Is\_Vet **IS** 'Oracle does not support BOOLEAN attributes, this is a pseudoBoolean to create a flag, currently unknown if the programmers will use upper or lower case, or a 1|0 to set the flag have taken all into account';

**CREATE** **TABLE** "DAVEDBA"."VET\_PROCEDURE"

(    "VET\_PROCEDUREID" int GENERATED **AS** IDENTITY **PRIMARY KEY**,

    "SPECIALITYID" int **DEFAULT** NULL,

    "VET\_PROCEDURE\_NAME" *VARCHAR2*(30 BYTE),

    "IS\_SURGERY" *CHAR*(1 BYTE),

    "VET\_PROCEDURE\_COST" *NUMBER*(7,2)

) SEGMENT CREATION DEFERRED

**PCTFREE** 10 **PCTUSED** 40 INITRANS 1 **MAXTRANS** 255

NOCOMPRESS **LOGGING**

**TABLESPACE** "PERSONNEL" ;

**COMMENT** **ON** **COLUMN** "DAVEDBA"."VET\_PROCEDURE"."SPECIALITYID" **IS** 'Null is ok, especially if primary care';

**COMMENT** **ON** **TABLE** "DAVEDBA"."VET\_PROCEDURE" **IS** 'Originally named PROCEDURE during the creation of the ERD was changed to VET\_PROCEDURE to avoid confusion with the function "procedure''';

##### Manual alterations to the RX tables

**ALTER** **TABLE** RX\_ORDER

MODIFY DRUG\_UNITS\_PRESCRIBED *number*(9,2);

**ALTER** **TABLE** RX\_ORDER

MODIFY DRUG\_UNITS\_DISPENSED *number*(9,2);

**ALTER** **TABLE** RX\_ORDER

MODIFY DRUG\_UNITS\_DISPENSED *number*(9,2);

**ALTER** **TABLE** RX\_ORDER

MODIFY TIMES\_PER\_DAY *number*(9,2);

**ALTER** **TABLE** RX\_ORDER

**ADD** **COLUMN** CONTROLLED\_CHECKER *number*(9,2);

**ALTER** **TABLE** RX\_HISTORY

MODIFY DRUG\_UNITS\_PRESCRIBED *number*(9,2);

**ALTER** **TABLE** RX\_HISTORY

MODIFY DRUG\_UNITS\_DISPENSED *number*(9,2);

**ALTER** **TABLE** RX\_HISTORY

MODIFY DRUG\_UNITS\_DISPENSED *number*(9,2);

**ALTER** **TABLE** RX\_HISTORY

MODIFY TIMES\_PER\_DAY *number*(9,2);

**ALTER** **TABLE** RX\_ORDER **ADD** **CONSTRAINT** chk\_contr **CHECK** (CONTROLLED\_CHECKER **<=** 14);

##### Sequence Definitions

**CREATE** **SEQUENCE** breedid\_seq

MINVALUE 1

MAXVALUE 99999

**START WITH** 1

**INCREMENT BY** 1

NOCACHE;

**CREATE** **SEQUENCE** speciesid\_seq

MINVALUE 1

MAXVALUE 99999

**START WITH** 1

**INCREMENT BY** 1

NOCACHE;

**CREATE** **SEQUENCE** genderid\_seq

MINVALUE 1

MAXVALUE 20

**START WITH** 1

**INCREMENT BY** 1

NOCACHE;

**CREATE** **SEQUENCE** prescription\_seq

MINVALUE 1

**START WITH** 1000

**INCREMENT BY** 3

NOCACHE;

**ALTER** **TABLE** ANIMAL\_GENDER

MODIFY GENDERID int **DEFAULT** genderid\_seq**.NEXTVAL**;

**CREATE** **SEQUENCE** laborderid\_seq

MINVALUE 1

**START WITH** 1

**INCREMENT BY** 1

NOCACHE;

**CREATE** **SEQUENCE** estimate\_seq

MINVALUE 100

**START WITH** 100

**INCREMENT BY** 1

NOCACHE;

**CREATE** **SEQUENCE** invoice\_seq

**START WITH** 1000

**INCREMENT BY** 1

NOCACHE;

#### MATERIALIZED VIEWS

Note: some tables in screenshots are materialized views that Oracle chooses to display in two locations of the navigation tree.

##### CHART\_META\_MV & cHART\_META\_GROUPED\_SETS\_mv

**CREATE** **MATERIALIZED VIEW** CHART\_META\_MV

REFRESH COMPLETE **START WITH** (SYSDATE) NEXT (SYSDATE**+**1**/**1440) **WITH** *ROWID*

**AS**

**SELECT** **\***

**FROM**(

**SELECT** af.PETID, af.BIRTH\_DATE, af.BREEDID, af.GENDERID, af.SPECIESID, af.TEMPERAMENT\_NOTES, vph.VET\_PROCEDUREID, vph.VET\_PROCEDURE\_DATE, TO\_CHAR(vph.VET\_PROCEDURE\_NOTES) **as** VET\_PROCEDURE\_NOTES,

vph.VET\_PROCEDURE\_FOLLOWUP\_DATE, TO\_CHAR(vph.VET\_PROCEDURE\_FOLLOWUP\_OUTCOME) **as** PROCEDURE\_FOLLOWUP\_OUTCOME,

rank() **over** (**partition by** af.PETID **ORDER BY** vph.VET\_PROCEDURE\_DATE ) rnk,

ph.LABID, ph.DATE\_COMPLETED, TO\_CHAR(ph.RESULTS) **as** PROCEDURE\_RESULTS, ph.CRITICAL\_DISEASE,

rank() **over** (**partition by** af.PETID **ORDER BY** ph.DATE\_COMPLETED ) rnk2,

rxh.DRUGID, rxh.DRUG\_DOSAGE, rxh.DRUG\_UNITS\_PRESCRIBED, rxh.DATE\_WRITTEN, TO\_CHAR(rxh.NOTES) **AS** RX\_NOTES,

rank() **over** (**partition by** af.PETID **ORDER BY** rxh.DATE\_WRITTEN ) rnk3,

eh.ENCOUNTER\_WEIGHT, eh.ENCOUNTER\_DATE\_TIME, TO\_CHAR(eh.ENCOUNTER\_NOTES) **AS** ENCOUNTER\_NOTES,

rank() **over** (**partition by** af.PETID **ORDER BY** eh.ENCOUNTER\_DATE\_TIME ) rnk4,

rh.RADIMG\_DATE\_TAKEN, TO\_CHAR(rh.RADIMG\_NOTES) **as** RADIMG\_NOTES,

rank() **over** (**partition by** af.PETID **ORDER BY** rh.RADIMG\_DATE\_TAKEN ) rnk5

**FROM** ANIMAL\_FACTS af FULL OUTER **JOIN** VET\_PROCEDURE\_HISTORY vph

**ON** af.PETID **=** vph.PETID FULL OUTER **JOIN** PATHOLOGY\_HISTORY ph

**ON** af.PETID **=** ph.PETID FULL OUTER **JOIN** RX\_HISTORY rxh

**ON** af.PETID **=** rxh.PETID FULL OUTER **JOIN** ENCOUNTER\_HISTORY eh

**ON** af.PETID **=** eh.PETID FULL OUTER **JOIN** RADIOLOGY\_HISTORY rh

**ON** af.PETID **=** rh.PETID)

**WHERE** /\*rnk in (SELECT rank() over (partition by af.PETID ORDER BY vph.VET\_PROCEDURE\_DATE )

FROM ANIMAL\_FACTS af FULL OUTER JOIN VET\_PROCEDURE\_HISTORY vph

ON af.PETID = vph.PETID )

AND rnk2 IN (SELECT rank() over (partition by af.PETID ORDER BY ph.DATE\_COMPLETED )

FROM ANIMAL\_FACTS af FULL OUTER JOIN PATHOLOGY\_HISTORY ph

ON af.PETID = ph.PETID)

AND rnk3 IN (SELECT rank() over (partition by af.PETID ORDER BY rxh.DATE\_WRITTEN )

FROM ANIMAL\_FACTS af FULL OUTER JOIN RX\_HISTORY rxh

ON af.PETID = rxh.PETID)

AND \*/ rnk4 **IN** (**SELECT** rank() **over** (**partition by** af.PETID **ORDER BY** eh.ENCOUNTER\_DATE\_TIME )

**FROM** ANIMAL\_FACTS af FULL OUTER **JOIN** ENCOUNTER\_HISTORY eh

**ON** af.PETID **=** eh.PETID)

**AND** rnk5 **IN** (**SElECT** rank() **over** (**partition by** af.PETID **ORDER BY** rh.RADIMG\_DATE\_TAKEN )

**FROM** ANIMAL\_FACTS af FULL OUTER **JOIN** RADIOLOGY\_HISTORY rh

**ON** af.PETID **=** rh.PETID)

;

-----------------------------------------------

**CREATE** **MATERIALIZED VIEW** CHART\_META\_GROUPED\_SETS\_MV

REFRESH COMPLETE **START WITH** (SYSDATE) NEXT (SYSDATE**+**1**/**1440) **WITH** *ROWID*

**AS**

**SELECT** af.PETID, af.BIRTH\_DATE, af.BREEDID, af.GENDERID, af.SPECIESID, af.TEMPERAMENT\_NOTES, vph.VET\_PROCEDUREID, vph.VET\_PROCEDURE\_DATE, TO\_CHAR(vph.VET\_PROCEDURE\_NOTES) **as** VET\_PROCEDURE\_NOTES,

vph.VET\_PROCEDURE\_FOLLOWUP\_DATE, TO\_CHAR(vph.VET\_PROCEDURE\_FOLLOWUP\_OUTCOME) **as** PROCEDURE\_FOLLOWUP\_OUTCOME, ph.LABID, ph.DATE\_COMPLETED, TO\_CHAR(ph.RESULTS) **as** PROCEDURE\_RESULTS,

ph.CRITICAL\_DISEASE, rxh.DRUGID, rxh.DRUG\_DOSAGE, rxh.DRUG\_UNITS\_PRESCRIBED, rxh.DATE\_WRITTEN, TO\_CHAR(rxh.NOTES) **AS** RX\_NOTES, eh.ENCOUNTER\_WEIGHT, eh.ENCOUNTER\_DATE\_TIME, TO\_CHAR(eh.ENCOUNTER\_NOTES) **AS** ENCOUNTER\_NOTES, rh.RADIMG\_DATE\_TAKEN, TO\_CHAR(rh.RADIMG\_NOTES) **as** RADIMG\_NOTES

**FROM** ANIMAL\_FACTS af FULL OUTER **JOIN** VET\_PROCEDURE\_HISTORY vph

**ON** af.PETID **=** vph.PETID FULL OUTER **JOIN** PATHOLOGY\_HISTORY ph

**ON** af.PETID **=** ph.PETID FULL OUTER **JOIN** RX\_HISTORY rxh

**ON** af.PETID **=** rxh.PETID FULL OUTER **JOIN** ENCOUNTER\_HISTORY eh

**ON** af.PETID **=** eh.PETID FULL OUTER **JOIN** RADIOLOGY\_HISTORY rh

**ON** af.PETID **=** rh.PETID;

**GROUP BY** GROUPING SETS ( af.PETID, af.BIRTH\_DATE, af.BREEDID, af.GENDERID, af.SPECIESID, af.TEMPERAMENT\_NOTES, vph.VET\_PROCEDUREID, vph.VET\_PROCEDURE\_DATE, TO\_CHAR(vph.VET\_PROCEDURE\_NOTES),

vph.VET\_PROCEDURE\_FOLLOWUP\_DATE, TO\_CHAR(vph.VET\_PROCEDURE\_FOLLOWUP\_OUTCOME), ph.LABID, ph.DATE\_COMPLETED, TO\_CHAR(ph.RESULTS),

ph.CRITICAL\_DISEASE, rxh.DRUGID, rxh.DRUG\_DOSAGE, rxh.DRUG\_UNITS\_PRESCRIBED, rxh.DATE\_WRITTEN, TO\_CHAR(rxh.NOTES), eh.ENCOUNTER\_WEIGHT, eh.ENCOUNTER\_DATE\_TIME, TO\_CHAR(eh.ENCOUNTER\_NOTES), rh.RADIMG\_DATE\_TAKEN, TO\_CHAR(rh.RADIMG\_NOTES));

##### PATIENT\_CHECK\_IN\_MV

**CREATE** **MATERIALIZED VIEW** PATIENT\_CHECK\_IN\_MV

NOCOMPRESS **LOGGING**

**TABLESPACE** "CRM"

**PCTFREE** 10 **PCTUSED** 40

REFRESH COMPLETE

**START WITH** SYSDATE NEXT SYSDATE **+** 1**/**24

**AS**

**SELECT** o.OWNERID, p.PET\_FIRST\_NAME, p.PET\_MIDDLE\_NAME, o.LAST\_NAME, s.SPECIES\_NAME, ab.BREED\_NAME, RTRIM(FUNC\_PET\_SIBLINGS(PETID), ', ') **AS** "Patient's Animal Siblings"

**FROM** PET p **JOIN** OWNER o **ON**

        p.ownerid **=** o.ownerid **JOIN**

            ANIMAL\_SPECIES s **ON**

                p.SPECIESID **=** s.SPECIESID **JOIN**

                    ANIMAL\_BREED ab **ON**

                        p.BREEDID **=** ab.BREEDID

**WHERE** IS\_LIVING **IN** ('y', 'Y', '1'); --restricting to show only living pets so a deceased pet is not checked in.

##### PHARMA\_STOCK\_MV

**CREATE** **MATERIALIZED VIEW** PHARMA\_STOCK\_MV

REFRESH COMPLETE **START WITH** (SYSDATE) NEXT SYSDATE**+**1

**AS**

**SELECT** DRUGID, DRUG\_NAME, DRUG\_DOSAGE, (**CASE** **WHEN** REORDER\_FLAG **IN** ('1', 'y', 'Y', 't', 'T') **THEN** 'ORDER MORE' **ELSE** NULL **END**) **as** "Reorder?" ,

      DRUG\_UNITS\_INV, ORDER\_LEVEL, DATE\_STOCKED, DATE\_EXPIRATION

**FROM** PHARMACOLOGY\_STOCK;

#### VIEWS

##### CHART\_HEAD\_V

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

**AS**

**SELECT** PETID, CHART\_PKG.*FUNC\_CHART\_NAME*(PETID) **AS** "Animal Name", FUNC\_SPECIES(SPECIESID) **AS** "Species", FUNC\_BREED(BREEDID) **as** "Breed", FUNC\_GENDER(GENDERID) **as** "Gender", TO\_NUMBER(TRUNC(MONTHS\_BETWEEN(SYSDATE,BIRTH\_DATE)**/**12, 1)) **as** "Age", TEMPERAMENT\_NOTES **as** "Disposition"

**FROM** ANIMAL\_FACTS

**WHERE** DEATH\_DATE IS NULL;

##### CHART\_META\_V & CHART\_META\_GROUPED\_SETS\_V

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

**AS**

**SELECT** PETID, CHART\_PKG.*FUNC\_CHART\_NAME*(PETID) **AS** PATIENT\_NAME, FUNC\_OWNER\_NAME(PETID) **AS** OWNER\_NAME, FUNC\_ALL\_SIBLING\_BREEDS(PETID) **AS** PET\_SIBLINGS,

BIRTH\_DATE,

FUNC\_BREED(BREEDID) **AS** BREED,

FUNC\_GENDER(GENDERID) **AS** GENDER,

FUNC\_SPECIES(SPECIESID) **AS** SPECIES,

TEMPERAMENT\_NOTES,

FUNC\_PROCNAME(VET\_PROCEDUREID) **AS** CLINICAL\_PROCEDURE,

VET\_PROCEDURE\_DATE,

VET\_PROCEDURE\_NOTES,

VET\_PROCEDURE\_FOLLOWUP\_DATE,

PROCEDURE\_FOLLOWUP\_OUTCOME,

FUNC\_LAB\_NAME(LABID) **AS** PATHOLOGY\_LAB,

DATE\_COMPLETED,

PROCEDURE\_RESULTS,

CRITICAL\_DISEASE,

FUNC\_DRUGNAME(DRUGID) **AS** DRUG\_NAME,

DRUG\_DOSAGE,

DRUG\_UNITS\_PRESCRIBED,

DATE\_WRITTEN,

RX\_NOTES,

ENCOUNTER\_WEIGHT,

ENCOUNTER\_DATE\_TIME,

ENCOUNTER\_NOTES,

RADIMG\_DATE\_TAKEN,

RADIMG\_NOTES

**FROM** CHART\_META\_MV

**ORDER BY** PETID;

**CREATE OR REPLACE** **VIEW** CHART\_META\_GROUPED\_SETS\_V

**AS**

**SELECT** PETID, CHART\_PKG.*FUNC\_CHART\_NAME*(PETID) **AS** PATIENT\_NAME, FUNC\_OWNER\_NAME(PETID) **AS** OWNER\_NAME, FUNC\_ALL\_SIBLING\_BREEDS(PETID) **AS** PET\_SIBLINGS,

BIRTH\_DATE,

FUNC\_BREED(BREEDID) **AS** BREED,

FUNC\_GENDER(GENDERID) **AS** GENDER,

FUNC\_SPECIES(SPECIESID) **AS** SPECIES,

TEMPERAMENT\_NOTES,

FUNC\_PROCNAME(VET\_PROCEDUREID) **AS** CLINICAL\_PROCEDURE,

VET\_PROCEDURE\_DATE,

VET\_PROCEDURE\_NOTES,

VET\_PROCEDURE\_FOLLOWUP\_DATE,

PROCEDURE\_FOLLOWUP\_OUTCOME,

FUNC\_LAB\_NAME(LABID) **AS** PATHOLOGY\_LAB,

DATE\_COMPLETED,

PROCEDURE\_RESULTS,

CRITICAL\_DISEASE,

FUNC\_DRUGNAME(DRUGID) **AS** DRUG\_NAME,

DRUG\_DOSAGE,

DRUG\_UNITS\_PRESCRIBED,

DATE\_WRITTEN,

RX\_NOTES,

ENCOUNTER\_WEIGHT,

ENCOUNTER\_DATE\_TIME,

ENCOUNTER\_NOTES,

RADIMG\_DATE\_TAKEN,

RADIMG\_NOTES

**FROM** CHART\_META\_GROUPED\_SETS\_MV

**ORDER BY** PETID;

##### CHART\_NOTES

**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;

##### DAILY\_PROCEDURE\_RX

**CREATE OR REPLACE** **VIEW** DAILY\_PROCRX\_V

**AS**

**SELECT** vph.PETID, FUNC\_CHECK\_IN\_NAME(vph.PETID) **AS** "Pet Name", vph.VETID, FUNC\_VET\_NAME(vph.VETID) **AS** "Veterinarian", ro.DRUG\_UNITS\_PRESCRIBED, ro.DRUG\_UNITS\_DISPENSED, vph.VET\_PROCEDURE\_NOTES, ro.DRUGID, ro.RXID

**FROM** VET\_PROCEDURE\_HISTORY vph **JOIN** RX\_HISTORY rh

**ON** vph.PATIENT\_VET\_PROCEDUREID **=** rh.PATIENT\_VET\_PROCEDUREID

**JOIN** RX\_ORDER ro **ON**

  rh.RXID **=** ro.RXID

**WHERE** vph.RX\_DISPENSED\_DURING **IN** ('1', 'Y', 'y')

**AND** ro.FILLED\_BY IS NULL

**AND** TRUNC(vph.VET\_PROCEDURE\_DATE) **=** TRUNC(SYSDATE);

/\*REPORT SHOWS WHAT PROCEDURES WERE DONE THAT DAY WITH MEDICINE

DISPENSED DURING THE PROCEDURE SO THE CHARTS CAN BE

UPDATED BY PHARMACISTS/VETS BEFORE CLOSE OF BUSINESS

This is both common sense and a regulation, when you've got

an animal crashing and you need to administer meds quickly you

don't want to have to write it down that second\*/

##### FULL\_CHARTS

--PART 1

**CREATE OR REPLACE** **VIEW** FULL\_TEXT\_CHART\_DECEASED\_V

**AS**

**SELECT** PETID, CHART\_PKG.*FUNC\_FLTXTCHART*(PETID) **AS** CHART

**FROM** ANIMAL\_FACTS

**WHERE** DEATH\_DATE IS NOT NULL

**ORDER BY** PETID;

--PART 2

**CREATE OR REPLACE** **FORCE** EDITIONABLE **VIEW** "DAVEDBA"."FULL\_TEXT\_CHART\_LIVING\_V" ("PETID", "CHART") **AS**

**SELECT** PETID, CHART\_PKG.*FUNC\_FLTXTCHART*(PETID) **AS** CHART

**FROM** ANIMAL\_FACTS

**WHERE** DEATH\_DATE IS NULL

**ORDER BY** PETID;

--PART 3

**CREATE OR REPLACE** **VIEW** SPLIT\_TEXT\_CHART\_LVNG\_V

**AS**

**SELECT** PETID, CHART\_PKG.*FUNC\_CHART\_NAME*(PETID) **AS** PET\_NAME, CHART\_PKG.*FUNC\_CHARTHEAD*(PETID) **AS** CHART\_HEAD, CHART\_PKG.*FUNC\_FULLCHARTNOTES*(PETID) **AS** CHART\_NOTES, CHART\_PKG.*FUNC\_RX\_CHART\_DETAILS*(PETID) **AS** RX\_CHART

**FROM** ANIMAL\_FACTS

**ORDER BY** PETID;

##### GRIEF\_COUNSELOR\_ADOPTION\_V

**CREATE OR REPLACE** **VIEW** GRIEF\_COUNSELOR\_ADOPTION\_V

**AS**

**SELECT** ALERTID, ALERT\_DATE, gca.PETID, FUNC\_CHECK\_IN\_NAME(gca.PETID) **as** "Deceased Name", DEATH\_DATE **AS** "Date of Passing", ENCOUNTER\_NOTES **as** "Method of Passing", OWNERID, FUNC\_OWNER\_NAME(gca.PETID) **as** "Parent Name", PHONE\_PRIMARY **as** "Parent Phone", FUNC\_ALL\_OTHER\_PETS(gca.PETID) **AS** "Previously owned pets", COMPLETE\_DATE, RESOLUTION\_NOTES

**FROM** GRIEF\_COUNSELOR\_ALERT gca **JOIN** ENCOUNTER\_HISTORY eh

**ON** gca.PETID **=** eh.PETID

**WHERE** COMPLETE\_DATE IS NULL

**AND** TRUNC( TO\_DATE( REGEXP\_SUBSTR(ENCOUNTER\_NOTES, '([0-9][0-9]\-[A-Z][A-Z][A-Z]\-[0-9][0-9])'), 'DD-MON-YY')) **>=** TRUNC(DEATH\_DATE);

##### MISC\_CHART\_VIEWS

--lab work done

**CREATE OR REPLACE** **VIEW** LAB\_WORK\_V

**AS**

**SELECT** PETID, PETID,

CHART\_PKG.*FUNC\_CHART\_NAME*(PETID) **AS** PET\_NAME, FUNC\_LAB\_NAME(LABID) **as** LAB\_NAME, RESULTS, DATE\_COMPLETED, CRITICAL\_DISEASE

**FROM** PATHOLOGY\_HISTORY ;

--last 5 years of maint medication

**CREATE OR REPLACE** **VIEW** RX\_HISTORY5YRS\_ALLMAINT\_MEDS\_V

**AS**

**SELECT** PETID, FUNC\_DRUGNAME(DRUGID) **as** DRUG\_NAME, DRUG\_DOSAGE, DATE\_WRITTEN, IS\_MAINTENANCE\_MED

**FROM** RX\_HISTORY

**WHERE** IS\_MAINTENANCE\_MED **IN** ('Y', 'y', '1')

**OR** DATE\_WRITTEN **>=** ( ADD\_MONTHS(TRUNC(DATE\_WRITTEN), **-**5**\***12));

--clinical procedure history

**CREATE OR REPLACE** **VIEW** PROCEDURE\_HISTORY\_V

**AS**

**SELECT** VET\_PROCEDUREID,

FUNC\_PROCNAME(VET\_PROCEDUREID) **AS** CLINICAL\_PROCEDURE,

PETID,

CHART\_PKG.*FUNC\_CHART\_NAME*(PETID) **AS** PET\_NAME,

VET\_PROCEDURE\_DATE,

VET\_PROCEDURE\_NOTES,

VET\_PROCEDURE\_FOLLOWUP\_DATE,

VET\_PROCEDURE\_FOLLOWUP\_OUTCOME

**FROM** VET\_PROCEDURE\_HISTORY ;

--radiology chart view

**CREATE OR REPLACE** **VIEW** RADIOLOGY\_V

**AS**

**SELECT** PETID, CHART\_PKG.*FUNC\_CHART\_NAME*(PETID) **AS** PET\_NAME,

RADIMG\_DATE\_TAKEN **AS** IMG\_DATE, RADIMG\_NOTES, RADIMG\_FILES, RADIMGID

**FROM** RADIOLOGY\_HISTORY;

##### PATH\_LABS\_OPEN

**CREATE OR REPLACE** **VIEW** PATH\_LABS\_OPEN\_V

**AS**

**SELECT** LABORDERID, LABID, FUNC\_LAB\_NAME(LABID) **AS** "Lab Name", FUNC\_CHECK\_IN\_NAME(PETID) **AS** "For", PETID **AS** "Patient Chart #", KITS\_ON\_HAND **AS** "Availible Kits"

**FROM** PATHOLOGY\_LAB\_ORDERS **JOIN** PATHOLOGY\_LAB\_TESTS **USING**(LABID)

**WHERE** TRUNC(DATE\_ORDERED) **=** TRUNC(SYSDATE) **AND** DATE\_COMPLETED IS NULL;

##### RX\_DETAILS\_V

**CREATE OR REPLACE** **VIEW** RX\_DETAILS\_V **AS**

**SELECT** rh.RXID, rh.PETID, CHART\_PKG.*FUNC\_CHART\_NAME*(rh.PETID) **AS** PET\_NAME, FUNC\_VET\_NAME(rh.VETID) **AS** VET\_NAME, rh.DRUGID, ps.DRUG\_NAME, rh.DRUG\_DOSAGE, rh.IS\_MAINTENANCE\_MED, rh.DRUG\_UNITS\_PRESCRIBED, rh.TIMES\_PER\_DAY, rh.DATE\_WRITTEN, rh.NOTES, rh.PATIENT\_VET\_PROCEDUREID, FUNC\_PROCNAME(vph.VET\_PROCEDUREID) **AS** CLINICAL\_EVENT

**FROM** RX\_HISTORY rh **JOIN** PHARMACOLOGY\_STOCK ps

**ON** rh.DRUGID **=** ps.DRUGID

FULL OUTER **JOIN** VET\_PROCEDURE\_HISTORY vph

**ON** rh.PATIENT\_VET\_PROCEDUREID **=** vph.PATIENT\_VET\_PROCEDUREID

**WHERE** (rh.DRUGID, ps.DRUG\_NAME) **IN** (**SELECT** DRUGID, DRUG\_NAME **FROM** PHARMACOLOGY\_STOCK)

**AND** ( rh.PATIENT\_VET\_PROCEDUREID **IN** (**SELECT** **DISTINCT** PATIENT\_VET\_PROCEDUREID **FROM** VET\_PROCEDURE\_HISTORY) **OR** rh.PATIENT\_VET\_PROCEDUREID IS NULL)

;

##### RX\_FROM\_PROCEDURES\_V

**CREATE OR REPLACE** **VIEW** RX\_FROM\_PROCEDURES\_V

**AS**

**SELECT** RXID, FUNC\_VET\_NAME(VETID) **AS** "Written by", vp.VET\_PROCEDURE\_NAME **AS** "Clinical Procedure", FUNC\_CHECK\_IN\_NAME(PETID) **AS** "For", FUNC\_OWNER\_NAME(petid) **AS** "OF", ro.DRUGID **AS** "Drug Stock #", po.DRUG\_NAME **AS** "Drug", DRUG\_UNITS\_PRESCRIBED **AS** "AMOUNT", ro.TIMES\_PER\_DAY **AS** "Times Per Day", ro.NUM\_REFILLS\_LEFT **AS** "REFILLS LEFT", ro.DATE\_SUBMITTED **AS** "Written on", ro.DATE\_FILLED, ro.FILLED\_BY

**FROM** RX\_ORDER ro **JOIN** PHARMACOLOGY\_STOCK po

**ON** ro.DRUGID **=** po.DRUGID **JOIN**

VET\_PROCEDURE vp

**ON** ro.VET\_PROCEDUREID **=** vp.VET\_PROCEDUREID

**WHERE** ro.FILLED\_BY IS NULL

**AND** ro.DATE\_FILLED IS NULL

**AND** (ro.VET\_PROCEDUREID, vp.VET\_PROCEDURE\_NAME) **IN** (**SELECT** VET\_PROCEDUREID, VET\_PROCEDURE\_NAME **FROM** VET\_PROCEDURE);

##### RX\_ORDERS\_TOFILL\_V

**CREATE OR REPLACE** **VIEW** RX\_ORDERS\_TOFILL\_V

**AS**

**SELECT** RXID, FUNC\_VET\_NAME(VETID) **AS** "Written by", FUNC\_CHECK\_IN\_NAME(PETID) **AS** "For", FUNC\_OWNER\_NAME(petid) **AS** "OF", ro.DRUGID **AS** "Drug Stock #", po.DRUG\_NAME **AS** "Drug", DRUG\_UNITS\_PRESCRIBED **AS** "AMOUNT", ro.TIMES\_PER\_DAY **AS** "Times Per Day", ro.NUM\_REFILLS\_LEFT **AS** "REFILLS LEFT", ro.DATE\_SUBMITTED **AS** "Written on", po.DRUG\_UNITS\_INV **AS** "On hand stock", FUNC\_RX\_SPECIES\_SAFE(ro.RXID) **AS** "Safe to Dispense", ro.DATE\_FILLED, ro.FILLED\_BY

**FROM** RX\_ORDER ro **JOIN** PHARMACOLOGY\_STOCK po

**ON** ro.DRUGID **=** po.DRUGID

**WHERE** ro.FILLED\_BY IS NULL

**AND** ro.DATE\_FILLED IS NULL

**AND** ro.VET\_PROCEDUREID IS NULL;

##### VARIOUS CHEM VIEWS

--PHARMACOLOGY IN STOCK

**CREATE OR REPLACE** **VIEW** PHARMACOLOGY\_ON\_HAND\_V

**AS**

**SELECT** DRUGID, DRUG\_NAME, DRUG\_DOSAGE, DRUG\_UNITS\_INV, IS\_CONTROLLED, DATE\_STOCKED, DATE\_EXPIRATION

**FROM** PHARMACOLOGY\_STOCK ;

--REPORT ON BLOOD ON HAND

**CREATE OR REPLACE** **VIEW** BLOOD\_REPORT\_V

**AS**

**SELECT** COUNT(BLOODBAGID) **AS** TOTAL\_BAGS, NVL(FUNC\_SPECIES(SPECIESID), 'Grand Total') **AS** SPECIES

**FROM** LOCAL\_BLOOD\_BANK

**GROUP BY** **CUBE** (SPECIESID);

##### INVOICE & ESTIMATE VIEWS

**CREATE OR REPLACE** **VIEW** ESTIMATE\_V

**AS**

**select** e.ESTIMATEID, FUNC\_CHECK\_IN\_NAME(e.PETID) **AS** PET\_NAME, e.DATE\_ESTIMATE\_CREATION, FUNC\_DRUGNAME(DRUGID) **AS** RX, RX\_COST, FUNC\_LAB\_NAME(LABID) **AS** LAB, LAB\_COST, SPECIALTY\_ADD\_ON\_COST, SPECIALTYID, FUNC\_PROCNAME( VET\_PROCEDUREID) **AS** CLINICAL\_PROCEDURE, PROCEDURE\_COST, ADD\_ONS\_DESCRIPTION, INDIVIDUAL\_ADD\_ON\_COST, OR\_FEE, LINE\_SUBTOTAL, grand\_total

**from** estimate e

**join** (**select** q.estimateid, sum(q.LINE\_SUBTOTAL) **as** grand\_total **from** estimate q **group by** q.estimateid) j **on** e.estimateid **=** j.estimateid;

**CREATE OR REPLACE** **FORCE** EDITIONABLE **VIEW** CUSTOMER\_INVOICE\_V **AS**

**SELECT** io.INVOICEID, FUNC\_CHECK\_IN\_NAME(io.PETID) **AS** PET\_NAME, io.DATE\_INVOICE\_CREATION, FUNC\_DRUGNAME(DRUGID) **AS** RX, RX\_COST,

     FUNC\_LAB\_NAME(LABID) **AS** LAB, LAB\_COST, SPECIALTY\_ADD\_ON\_COST, SPECIALTYID, FUNC\_PROCNAME( VET\_PROCEDUREID) **AS** CLINICAL\_PROCEDURE,

     PROCEDURE\_COST, ADD\_ONS\_DESCRIPTION, INDIVIDUAL\_ADD\_ON\_COST, OR\_FEE, LINE\_SUBTOTAL, LATE\_FEE, GRAND\_TOTAL **AS** LINE\_SUB\_WITH\_FEE, invoice\_total **from** INVOICE\_OPEN io

**JOIN** (**select** q.invoiceid, sum(q.GRAND\_TOTAL)**as** INVOICE\_TOTAL

**from** INVOICE\_OPEN q

**group by** q.INVOICEID) j

**ON** io.INVOICEID **=** j.INVOICEID;

### PROGRAM UNITS

#### Triggers—unshown triggers are ones automatically generated by Oracle

##### GREIF\_ALERT\_TRG

**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**;

##### IS\_VET\_TRG

**CREATE OR REPLACE** **TRIGGER** IS\_VET\_TRG

**AFTER** **INSERT** **ON** STAFF

**FOR** EACH **ROW**

**DECLARE**

**PRAGMA AUTONOMOUS\_TRANSACTION**;

lv\_vetid int;

lv\_flag *char*;

**BEGIN**

**SELECT** MAX(StaffID)

**INTO** lv\_vetid

**FROM** STAFF;

**SELECT** IS\_VET

**INTO** lv\_flag

**FROM** STAFF

**WHERE** staffid **=** lv\_vetid;

**IF** lv\_flag **=** 1 **OR** lv\_flag **=** 'y' **OR** lv\_flag **=** 'Y'

**THEN** **INSERT** **INTO** VETERINARIAN(VETID) **VALUES**(lv\_vetid);

**END IF**;

COMMIT;

**END**;

##### PET\_FACTS\_TRIG

**CREATE OR REPLACE** **TRIGGER** PET\_FACTS\_TRG

**AFTER** **INSERT** **ON** PET

**DECLARE**

lv\_petid int;

--using max PETID as there will never be an insert of a smaller ID and this is a way of getting around a VERY long series of steps

**BEGIN**

**SELECT** MAX(PETID)

**INTO** lv\_petid

**FROM** PET;

**INSERT** **INTO** ANIMAL\_FACTS

(PETID, PET\_FIRST\_NAME, PET\_MIDDLE\_NAME, OWNER\_LAST\_NAME, SPECIESID, BREEDID, GENDERID, COLORING, BIRTH\_DATE, TEMPERAMENT\_NOTES)

(**SELECT** PetID, PET\_FIRST\_NAME, PET\_MIDDLE\_NAME, o.Last\_Name, SPECIESID, BREEDID, GENDERID, COLORING, BIRTH\_DATE, TEMPERAMENT\_NOTES

**FROM** PET p **JOIN** OWNER o **USING**(OwnerID)

**WHERE** p.petid **=** lv\_petid

);

**END**;

##### PET\_FACTS\_UPDATE\_TRG

**create or replace** **TRIGGER** PET\_FACTS\_UPDATE\_TRIG

**AFTER** **UPDATE** **OF** BIRTH\_DATE,BREEDID,COLORING,GENDERID, OWNERID, PET\_FIRST\_NAME, PET\_MIDDLE\_NAME, SPECIESID, TEMPERAMENT\_NOTES **ON** PET

**DECLARE**

lv\_petid int;

**BEGIN**

/\*

We are using a "WHERE EXISTS" because we want the statement to find all rows that exist in in the table we are checking

and update them based on the values of the table that we have UPDATED \*/

**UPDATE** ANIMAL\_FACTS

**SET** ( BIRTH\_DATE,BREEDID,COLORING,GENDERID,PET\_FIRST\_NAME, PET\_MIDDLE\_NAME, SPECIESID, TEMPERAMENT\_NOTES)

**=** ( **SELECT** BIRTH\_DATE,BREEDID,COLORING,GENDERID, PET\_FIRST\_NAME,PET\_MIDDLE\_NAME,SPECIESID,TEMPERAMENT\_NOTES

**FROM** PET

**WHERE** ANIMAL\_FACTS.PETID **=** PET.PETID)

**WHERE** **EXISTS** ( **SELECT** BIRTH\_DATE,BREEDID,COLORING,GENDERID, PET\_FIRST\_NAME,PET\_MIDDLE\_NAME,SPECIESID,TEMPERAMENT\_NOTES

**FROM** PET

**WHERE** ANIMAL\_FACTS.PETID **=** PET.PETID);

**END**;

#### Functions

##### FUNC\_ALL\_OTHER\_PETS

**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**;

##### FUNC\_ALL\_SIBLING\_BREEDS

**CREATE OR REPLACE** FUNCTION FUNC\_ALL\_SIBLING\_BREEDS(

    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;

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

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

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

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

**WHERE** ab.BREEDID

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

**FROM** PET

**WHERE** PETID **<>** f\_petid

**AND** OWNERID **=** lv\_ownerid

**AND** IS\_LIVING **IN** ('1', 'y', 'Y'));

**BEGIN**

**SELECT** OWNERID

**INTO** lv\_ownerid

**FROM** PET

**WHERE** PETID **=** f\_petid;

**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**||**', ';

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

            lv\_caninecount **:=** lv\_caninecount **+** 1;

            lv\_canine**:=** lv\_canine **||** rec\_otherpets.BREED\_NAME**||**', ';

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

            lv\_felinecount **:=** lv\_felinecount **+** 1;

            lv\_feline**:=** lv\_feline **||** rec\_otherpets.BREED\_NAME**||**', ';

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

            lv\_reptilecount **:=** lv\_reptilecount **+** 1;

            lv\_reptile**:=** lv\_reptile **||** rec\_otherpets.BREED\_NAME**||**', ';

**ELSE** NULL;

**END CASE**;

    lv\_loop **:=** lv\_loop **+** 1;

**END LOOP**;

**CASE**

**WHEN** lv\_loop IS NULL **OR** lv\_loop **=** 1 **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 **||** lv\_aviancount**||**' birds of the following breeds: '**||**lv\_avian**||**' '**||** CHR(10);

**END IF**;

**IF** lv\_caninecount **>** 0

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

**END IF**;

**IF** lv\_felinecount **>** 0

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

**END IF**;

**IF** lv\_reptilecount **>** 0

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

**END IF**;

        lv\_other\_pets **:=** lv\_other\_pets**||**'End of other known animal siblings for this patient.';

**END CASE**;

**RETURN** lv\_other\_pets;

**END**;

##### FUNC\_BREED

**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**;

##### FUNC\_CHECK\_IN\_NAME

**CREATE OR REPLACE** FUNCTION FUNC\_CHECK\_IN\_NAME (in\_petid **IN** int)

**RETURN** *varchar2*

**AS**

--NO "DECLARE IN FUNCTIONS JUST STUFF IT IN AFTER AS"

--USING ANCHORED DATA TYPES TO AVOID ERRORS

    lv\_petfirst PET.PET\_FIRST\_NAME%**TYPE**;

    lv\_petmid PET.PET\_MIDDLE\_NAME%**TYPE**;

    lv\_petlast OWNER.LAST\_NAME%**TYPE**;

    lv\_full\_name *varchar2*(200);

**BEGIN**

**SELECT** p.PET\_FIRST\_NAME, p.PET\_MIDDLE\_NAME, o.LAST\_NAME

**INTO** lv\_petfirst, lv\_petmid, lv\_petlast

**FROM** PET p **JOIN** OWNER o **USING**(OWNERID)

**WHERE** P.PETID **=** in\_petid;

    lv\_full\_name **:=** lv\_petfirst **||**' ' **||** lv\_petmid **||**' '**||** lv\_petlast;

**RETURN** lv\_full\_name;

**EXCEPTION**

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

    DBMS\_OUTPUT.*PUT\_LINE*('This animal id is invalid, are you sure you entered it in correctly?');

**END**;

##### FUNC\_DRUGNAME

**CREATE OR REPLACE** FUNCTION FUNC\_DRUGNAME(f\_drugid **IN** PHARMACOLOGY\_STOCK.DRUGID%**TYPE**)

**RETURN** *VARCHAR2*

**AS**

lv\_drugname *varchar2*(2000) **:=** NULL;

**BEGIN**

**SELECT** DRUG\_NAME

**INTO** lv\_drugname

**FROM** PHARMACOLOGY\_STOCK

**WHERE** DRUGID **=** f\_drugid;

**RETURN** lv\_drugname;

**EXCEPTION**

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

**RETURN** NULL;

**END**;

##### FUNC\_DUALCLOB

**CREATE OR REPLACE** FUNCTION FUNC\_DUALCLOB (f\_value **IN** *varchar2*)

**RETURN** *CLOB*

**AS**

lv\_valueout *CLOB*;

**BEGIN**

**SELECT** f\_value

**INTO** lv\_valueout

**FROM** DUAL;

**RETURN** lv\_valueout;

**END**;

##### FUNC\_GENDER

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

    f\_genderid **IN** ANIMAL\_GENDER.GENDERID%**TYPE**)

**RETURN** *VARCHAR2*

**AS**

lv\_gender\_name ANIMAL\_GENDER.GENDER\_NAME%**TYPE**;

**BEGIN**

**SELECT** GENDER\_NAME

**INTO** lv\_gender\_name

**FROM** ANIMAL\_GENDER

**WHERE** GENDERID **=** f\_genderid;

**RETURN** lv\_gender\_name;

**EXCEPTION**

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

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

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

**RETURN** NULL;

**END**;

##### FUNC\_LAB\_NAME

**CREATE OR REPLACE** FUNCTION FUNC\_LAB\_NAME (f\_labid **IN** int)

**RETURN** *varchar2*

**AS**

lv\_labname PATHOLOGY\_LAB\_TESTS.LAB\_NAME%**TYPE**;

**BEGIN**

**SELECT** LAB\_NAME

**INTO** lv\_labname

**FROM** PATHOLOGY\_LAB\_TESTS

**WHERE** LABID **=** f\_labid;

**RETURN** lv\_labname;

**EXCEPTION**

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

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

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

**RETURN** NULL;

**END**;

##### FUNC\_ONWER\_NAME

**CREATE OR REPLACE** FUNCTION FUNC\_OWNER\_NAME (f\_petid **IN** int)

**RETURN** *varchar2*

**AS**

--USING ANCHORED DATA TYPES TO AVOID ERRORS

    lv\_ownerfirst OWNER.FIRST\_NAME%**TYPE**;

    lv\_petlast OWNER.LAST\_NAME%**TYPE**;

    lv\_full\_name *varchar2*(200);

**BEGIN**

**SELECT** o.FIRST\_NAME, o.LAST\_NAME

**INTO** lv\_ownerfirst, lv\_petlast

**FROM** PET p **JOIN** OWNER o **USING**(OWNERID)

**WHERE** P.PETID **=** in\_petid;

    lv\_full\_name **:=** lv\_ownerfirst **||**' '**||** lv\_petlast;

**RETURN** lv\_full\_name;

**EXCEPTION**

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

    DBMS\_OUTPUT.*PUT\_LINE*('This animal id is invalid, are you sure you entered it in correctly?' **||**chr(10)**||**

    'You are attempting to find the parent of an animal with a bad ID' );

**END**;

##### FUNC\_PET\_SIBLINGS

**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**;

##### FUNC\_PROCNAME

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

    f\_procid **IN** VET\_PROCEDURE\_HISTORY.VET\_PROCEDUREID%**TYPE**)

**RETURN** *varchar2*

**AS**

lv\_proc\_name VET\_PROCEDURE.VET\_PROCEDURE\_NAME%**TYPE**;

lv\_error *varchar2* **:=** 'none';

**BEGIN**

**SELECT** VET\_PROCEDURE\_NAME

**INTO** lv\_proc\_name

**FROM** VET\_PROCEDURE

**WHERE** VET\_PROCEDUREID**=** f\_procid;

**RETURN** lv\_proc\_name;

**EXCEPTION**

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

**RETURN** lv\_error;

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

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

**END**;

##### FUNC\_RX\_APPROVAL

**CREATE OR REPLACE** FUNCTION FUNC\_RX\_APPROVAL

(IN\_amt *NUMBER*, IN\_timesperday *NUMBER*)

**RETURN** *NUMBER*

**AS**

/\*This is a subfunction called by other functions to make sure that

the check constraint for controlled drugs is not violated\*/

lv\_daysfilled *NUMBER*(9,2);

**BEGIN**

    lv\_daysfilled **:=** (IN\_amt**/**IN\_timesperday);

**RETURN** lv\_daysfilled;

**END**;

##### FUNC\_RX\_SPECIES\_SAFE

**CREATE OR REPLACE** FUNCTION FUNC\_RX\_SPECIES\_SAFE(f\_rxid **IN** int)

**RETURN** *varchar2*

**AS**

    ex\_warning **EXCEPTION**;

    lv\_warning *varchar2*(100) **:=** 'WARNING: Drug is not safe for species; verify Rx with vet!';

    lv\_safe *varchar2*(100) **:=** 'Safe to dispense to this species';

    lv\_speciesid int **:=** NULL;

    lv\_flag\_avian *char*(1) **:=** NULL;

    lv\_flag\_canine *char*(1) **:=** NULL;

    lv\_flag\_feline *char*(1) **:=** NULL;

    lv\_flag\_reptile *char*(1) **:=** NULL;

    lv\_drugid int **:=** NULL;

    lv\_petid int **:=** NULL;

**BEGIN**

**SELECT** DRUGID, PETID

**INTO** lv\_drugid, lv\_petid

**FROM** RX\_ORDER

**WHERE** RXID **=** f\_rxid;

**SELECT** SPECIESID

**INTO** lv\_speciesid

**FROM** ANIMAL\_FACTS

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

**SELECT** AVIAN\_SAFE, CANINE\_SAFE, FELINE\_SAFE, REPTILE\_SAFE

**INTO** lv\_flag\_avian, lv\_flag\_canine, lv\_flag\_feline, lv\_flag\_reptile

**FROM** PHARMACOLOGY\_STOCK

**WHERE** DRUGID **=** lv\_drugid;

/\*AVIAN SPECIESID = 1, CANINE =2, FELINE = 3, REPTILE = 4 \*/

**CASE**

**WHEN** lv\_speciesid **=** 1 **AND** lv\_flag\_avian **IN** ('Y', 'y', 'T', 't', '1') **THEN**

         lv\_safe **:=** 'Safe to dispense to this bird';

**WHEN** lv\_speciesid **=** 2 **AND** lv\_flag\_canine **IN** ('Y', 'y', 'T', 't', '1') **THEN**

        lv\_safe **:=** 'Safe to dispense to this dog';

**WHEN** lv\_speciesid **=** 3 **AND** lv\_flag\_feline **IN** ('Y', 'y', 'T', 't', '1') **THEN**

         lv\_safe **:=** 'Safe to dispense to this cat';

**WHEN** lv\_speciesid **=** 4 **AND** lv\_flag\_reptile **IN** ('Y', 'y', 'T', 't', '1') **THEN**

         lv\_safe **:=** 'Safe to dispense to this reptile';

**ELSE** RAISE ex\_warning;

**END CASE**;

**RETURN** lv\_safe;

**EXCEPTION**

**WHEN** ex\_warning **THEN**

**RETURN** lv\_warning;

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

    lv\_warning **:=** 'no data found';

**RETURN** lv\_warning;

**END**;

##### FUNC\_SPECIES

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

f\_speciesid **IN** ANIMAL\_SPECIES.SPECIESID%**TYPE**)

**RETURN** *varchar2*

**AS**

    lv\_species\_name ANIMAL\_SPECIES.SPECIES\_NAME%**TYPE**;

**BEGIN**

**SELECT** SPECIES\_NAME

**INTO** lv\_species\_name

**FROM** ANIMAL\_SPECIES

**WHERE** SPECIESID **=** f\_speciesid;

**RETURN** lv\_species\_name;

**EXCEPTION**

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

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

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

**RETURN** NULL;

**END**;

##### FUNC\_VET\_NAME

**create or replace** FUNCTION FUNC\_VET\_NAME (in\_vetid **IN** int)

**RETURN** *varchar2*

**AS**

--NO "DECLARE IN FUNCTIONS JUST STUFF IT IN AFTER AS"

--USING ANCHORED DATA TYPES TO AVOID ERRORS

    lv\_vetfirst STAFF.STAFF\_FIRST\_NAME%**TYPE**;

    lv\_vetlast STAFF.STAFF\_LAST\_NAME%**TYPE**;

    lv\_full\_name *varchar2*(200);

**BEGIN**

**SELECT** s.STAFF\_FIRST\_NAME, s.STAFF\_LAST\_NAME

**INTO** lv\_vetfirst, lv\_vetlast

**FROM** VETERINARIAN v **JOIN** STAFF s

**ON** v.VETID **=** s.STAFFID

**WHERE** V.VETID **=** in\_vetid;

    lv\_full\_name **:=** lv\_vetfirst **||**' '**||** lv\_vetlast;

**RETURN** lv\_full\_name;

**EXCEPTION**

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

    DBMS\_OUTPUT.*PUT\_LINE*('This veterinarian id, are you sure you entered it in correctly?' );

**END**;

##### FUNC\_VET\_STAFF

**CREATE OR REPLACE** FUNCTION FUNC\_VET\_STAFF (in\_vetid **IN** int)

**RETURN** *varchar2*

**AS**

--USING ANCHORED DATA TYPES TO AVOID ERRORS

  lv\_vetfirst STAFF.STAFF\_FIRST\_NAME%**TYPE**;

  lv\_vetlast STAFF.STAFF\_LAST\_NAME%**TYPE**;

  lv\_full\_name *varchar2*(200);

lv\_staff *varchar2*(1000) **:=** 'Staff Encounter';

**BEGIN**

**SELECT** s.STAFF\_FIRST\_NAME, s.STAFF\_LAST\_NAME

**INTO** lv\_vetfirst, lv\_vetlast

**FROM** VETERINARIAN v **JOIN** STAFF s

**ON** v.VETID **=** s.STAFFID

**WHERE** V.VETID **=** in\_vetid;

  lv\_full\_name **:=** lv\_vetfirst **||**' '**||** lv\_vetlast;

**RETURN** lv\_full\_name;

**EXCEPTION**

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

**RETURN** lv\_staff;

  DBMS\_OUTPUT.*PUT\_LINE*('This staff id, are you sure you entered it in correctly?' );

**END**;

#### Procedures

##### PROC\_ADD\_STAFF

**CREATE OR REPLACE** PROCEDURE PROC\_ADD\_STAFF(

pv\_first **IN** *varchar2*,

pv\_last **IN** *varchar2*,

pv\_hiredate **IN** *date*,

pv\_isvet **IN** *char*

)

**AS**

    lv\_staffid int;

**BEGIN**

**INSERT** **INTO** STAFF (STAFF\_FIRST\_NAME, STAFF\_LAST\_NAME, EMPLOYMENT\_DATE, IS\_VET)

**VALUES**(pv\_first, pv\_last, pv\_hiredate, pv\_isvet)

**RETURNING** STAFFID **INTO** lv\_staffid;

    COMMIT;

**IF** pv\_isvet **=** 1 **OR** pv\_isvet **=** 'y' **OR** pv\_isvet **=** 'Y'

**THEN** **INSERT** **INTO** VETERINARIAN(VETID) **VALUES**(lv\_staffid);

      COMMIT;

**END IF**;

**END**;

##### PROC\_BULK\_RXPROC\_APPROVAL

**CREATE OR REPLACE** PROCEDURE PROC\_BULK\_RXPROC\_APPROVAL (

    p\_assent **IN** *char*,

    p\_approver **IN** int)

**AS**

**CURSOR** cur\_procrxs **IS**

**SELECT** PETID, VETID, DRUG\_UNITS\_PRESCRIBED, DRUGID, DRUG\_UNITS\_DISPENSED, RXID

**FROM** DAILY\_PROCRX\_V;

**TYPE** type\_rxupdate **IS** **RECORD**

    (

        pet DAILY\_PROCRX\_V.PETID%**TYPE**,

        vet DAILY\_PROCRX\_V.VETID%**TYPE**,

        drugrxd DAILY\_PROCRX\_V.DRUG\_UNITS\_PRESCRIBED%**TYPE**,

        drugdisp DAILY\_PROCRX\_V.DRUG\_UNITS\_DISPENSED%**TYPE**,

        drugid RX\_ORDER.DRUGID%**TYPE**,

        rxid1 RX\_ORDER.RXID%**TYPE**

    );

    rec\_rxupdate type\_rxupdate;

**BEGIN**

**IF** p\_assent **in** ('y','Y','1')

**THEN**

**OPEN** cur\_procrxs;

**LOOP**

**FETCH** cur\_procrxs **INTO** rec\_rxupdate;

**UPDATE** RX\_ORDER

**SET** DRUG\_UNITS\_DISPENSED **=** rec\_rxupdate.drugrxd, --they are affirming the amount dispensed are the same prescribed

           FILLED\_BY **=** p\_approver

           --DATE\_FILLED = SYSTIMESTAMP, nope....put this in for when it originally gets put into the system

**WHERE** RXID **=** rec\_rxupdate.rxid1;

**UPDATE** RX\_HISTORY

**SET** DATE\_FILLED **=** SYSDATE

**WHERE** RXID **=** rec\_rxupdate.rxid1;

          PROC\_DRG\_STOCK(rec\_rxupdate.drugid, rec\_rxupdate.drugrxd); --again by using this program they are affirming what was RX'd is what was filled.

**EXIT** **WHEN** cur\_procrxs%NOTFOUND;

**END LOOP**;

**END IF**;

**END**;

##### PROC\_CHART\_SEARCH

**CREATE OR REPLACE** PROCEDURE PROC\_CHART\_SEARCH (p\_petid **IN** int, p\_phrase1 **IN** *varchar2*, p\_phrase2 **in** *varchar2* **:=** NULL, p\_phrase3 *varchar2* **:=** NULL)

**AS**

cur\_chart1 sys\_refcursor;

cur\_chart2 sys\_refcursor;

cur\_chart3 sys\_refcursor;

results\_rec chart\_meta\_mv%**rowtype**;

**BEGIN**

DBMS\_OUTPUT.*PUT\_LINE* ('Search terms displaying below: not all attributes shown will have the searched term, these simply show what record has the term you seek.');

DBMS\_OUTPUT.*PUT\_LINE*('..............................................................................................................');

**IF** p\_phrase2 IS NULL **AND** p\_phrase3 IS NULL

**THEN**

**OPEN** cur\_chart1 **FOR**

**SELECT** **\*** **FROM** CHART\_META\_MV

**WHERE** PETID **=** p\_petid **AND**

                (VET\_PROCEDURE\_NOTES **LIKE** p\_phrase1 **OR**

                PROCEDURE\_FOLLOWUP\_OUTCOME **LIKE** '%' **||** p\_phrase1 **||** '%' **OR**

                RX\_NOTES **LIKE**'%' **||** p\_phrase1 **||** '%' **OR**

                ENCOUNTER\_NOTES **LIKE** '%' **||** p\_phrase1 **||** '%');

**LOOP**

**FETCH** cur\_chart1 **INTO** results\_rec;

    DBMS\_OUTPUT.*PUT\_LINE*('PETID:'**||**CHART\_PKG.*FUNC\_CHART\_NAME*(results\_rec.PETID)**||**

                        CHR(13)**||**' ENCOUNTER NOTES : ' **||** results\_rec.ENCOUNTER\_NOTES**||**

                        CHR(13)**||**' PROCEDURE NOTES : '**||** results\_rec.VET\_PROCEDURE\_NOTES**||**

                        CHR(13)**||**' FOLLOW UP OUTCOME: '**||** results\_rec.PROCEDURE\_FOLLOWUP\_OUTCOME**||**

                        CHR(13)**||**' PRESCRIPTION\_NOTES: '**||**results\_rec.RX\_NOTES**||**

                        CHR(13)**||**'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_');

**EXIT** **WHEN** cur\_chart1%NOTFOUND;

**END LOOP**;

**ELSIF** p\_phrase3 IS NULL

**THEN**

**OPEN** cur\_chart2 **FOR**

**SELECT** **\*** **FROM** CHART\_META\_MV

**WHERE** PETID **=** p\_petid **AND**

                (VET\_PROCEDURE\_NOTES **LIKE** p\_phrase1 **OR**

                PROCEDURE\_FOLLOWUP\_OUTCOME **LIKE** '%' **||** p\_phrase1 **||** '%' **OR**

                RX\_NOTES **LIKE**'%' **||** p\_phrase1 **||** '%' **OR**

                ENCOUNTER\_NOTES **LIKE** '%' **||** p\_phrase1 **||** '%' **OR**

                VET\_PROCEDURE\_NOTES **LIKE** p\_phrase2 **OR**

                PROCEDURE\_FOLLOWUP\_OUTCOME **LIKE** '%' **||** p\_phrase2 **||** '%' **OR**

                RX\_NOTES **LIKE**'%' **||** p\_phrase2 **||** '%' **OR**

                ENCOUNTER\_NOTES **LIKE** '%' **||** p\_phrase2 **||** '%');

**LOOP**

**FETCH** cur\_chart2 **INTO** results\_rec;

    DBMS\_OUTPUT.*PUT\_LINE*('PETID:'**||**CHART\_PKG.*FUNC\_CHART\_NAME*(results\_rec.PETID)**||**

                        CHR(13)**||**' ENCOUNTER NOTES : ' **||** results\_rec.ENCOUNTER\_NOTES**||**

                        CHR(13)**||**' PROCEDURE NOTES : '**||** results\_rec.VET\_PROCEDURE\_NOTES**||**

                        CHR(13)**||**' FOLLOW UP OUTCOME: '**||** results\_rec.PROCEDURE\_FOLLOWUP\_OUTCOME**||**

                        CHR(13)**||**' PRESCRIPTION\_NOTES: '**||**results\_rec.RX\_NOTES**||**

                        CHR(13)**||**'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_');

**EXIT** **WHEN** cur\_chart2%NOTFOUND;

**END LOOP**;

**ELSE**

**OPEN** cur\_chart3 **FOR**

**SELECT** **\*** **FROM** CHART\_META\_MV

**WHERE** PETID **=** p\_petid **AND**

                    (VET\_PROCEDURE\_NOTES **LIKE** p\_phrase1 **OR**

                    PROCEDURE\_FOLLOWUP\_OUTCOME **LIKE** '%' **||** p\_phrase1 **||** '%' **OR**

                    RX\_NOTES **LIKE**'%' **||** p\_phrase1 **||** '%' **OR**

                    ENCOUNTER\_NOTES **LIKE** '%' **||** p\_phrase1 **||** '%' **OR**

                    VET\_PROCEDURE\_NOTES **LIKE** p\_phrase2 **OR**

                    PROCEDURE\_FOLLOWUP\_OUTCOME **LIKE** '%' **||** p\_phrase2 **||** '%' **OR**

                    RX\_NOTES **LIKE**'%' **||** p\_phrase2 **||** '%' **OR**

                    ENCOUNTER\_NOTES **LIKE** '%' **||** p\_phrase2 **||** '%' **OR**

                    VET\_PROCEDURE\_NOTES **LIKE** p\_phrase3 **OR**

                    PROCEDURE\_FOLLOWUP\_OUTCOME **LIKE** '%' **||** p\_phrase3 **||** '%' **OR**

                    RX\_NOTES **LIKE**'%' **||** p\_phrase3 **||** '%' **OR**

                    ENCOUNTER\_NOTES **LIKE** '%' **||** p\_phrase3 **||** '%');

**LOOP**

**FETCH** cur\_chart3 **INTO** results\_rec;

        DBMS\_OUTPUT.*PUT\_LINE*('PETID:'**||**CHART\_PKG.*FUNC\_CHART\_NAME*(results\_rec.PETID)**||**

                            CHR(13)**||**' ENCOUNTER NOTES : ' **||** results\_rec.ENCOUNTER\_NOTES**||**

                            CHR(13)**||**' PROCEDURE NOTES : '**||** results\_rec.VET\_PROCEDURE\_NOTES**||**

                            CHR(13)**||**' FOLLOW UP OUTCOME: '**||** results\_rec.PROCEDURE\_FOLLOWUP\_OUTCOME**||**

                            CHR(13)**||**' PRESCRIPTION\_NOTES: '**||**results\_rec.RX\_NOTES**||**

                            CHR(13)**||**'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_');

**EXIT** **WHEN** cur\_chart3%NOTFOUND;

**END LOOP**;

**END IF**;

**END**;

##### PROC\_DEATH\_RECORD

**CREATE OR REPLACE** PROCEDURE PROC\_DEATH\_RECORD(

    p\_inoffice **IN** *char*,

    p\_petid **IN** PET.PETID%**TYPE**,

    p\_date **IN** ANIMAL\_FACTS.DEATH\_DATE%**TYPE**,

    p\_vetid **IN** VETERINARIAN.VETID%**TYPE** **DEFAULT** NULL,

    p\_notes **IN** ENCOUNTER\_HISTORY.ENCOUNTER\_NOTES%**TYPE** **DEFAULT** NULL

    )

**AS**

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

lv\_clobbuilder *clob*;

**BEGIN**

**UPDATE** ANIMAL\_FACTS

**SET** DEATH\_DATE **=** p\_date

**WHERE** PETID **=** p\_petid;

/\*Case below is regarding if the animal died in the office or not \*/

**CASE**

**WHEN** p\_inoffice **IN** ('Y', 'y', '1') **THEN**

        lv\_clobbuilder **:=** CONCAT(CONCAT(CONCAT('In office death recorded notes including cause of death follow: ', p\_notes), 'RECORDED: '), p\_date);

**INSERT** **INTO** ENCOUNTER\_HISTORY(PETID, VETID, ENCOUNTER\_DATE\_TIME, ENCOUNTER\_NOTES)

**VALUES**(p\_petid, p\_vetid, SYSTIMESTAMP, lv\_clobbuilder);

**ELSE**

            lv\_clobbuilder **:=** CONCAT(CONCAT(CONCAT('Owner alerted us of animal death on ', p\_date), 'Addtional notes including cause of death if known follows: '), p\_notes);

**INSERT** **INTO** ENCOUNTER\_HISTORY(PETID, VETID, ENCOUNTER\_DATE\_TIME, ENCOUNTER\_NOTES)

**VALUES**(p\_petid, NULL, SYSTIMESTAMP, lv\_clobbuilder);

**END CASE**;

**UPDATE** PET

**SET** IS\_LIVING **=** 'N'

**WHERE** PETID **=** p\_petid;

--TWO NULL VALUES RELATE TO DEATH DATE AND MOVED\_AWAY. MOVED\_AWAY IS NOT RELEVANT IN THIS CASE DEATH DATE WILL BE UPDATED AFTER THIS.

**INSERT** **INTO** PET\_HISTORICAL

**SELECT** PETID, OWNERID, PET\_FIRST\_NAME, PET\_MIDDLE\_NAME, SPECIESID, BREEDID, GENDERID, COLORING, BIRTH\_DATE, IS\_LIVING, PHOTO, TEMPERAMENT\_NOTES, NULL, NULL

**FROM** PET

**WHERE** PETID **=** p\_petid;

    COMMIT;

**UPDATE** PET\_HISTORICAL

**SET** DEATH\_DATE **=** p\_date

**WHERE** PETID **=** p\_petid;

COMMIT;

**END**;

**SELECT** eh.ENCOUNTERID, p.petid, FUNC\_CHECK\_IN\_NAME(af.petid), p.IS\_LIVING, af.DEATH\_DATE, eh.ENCOUNTER\_NOTES, eh.ENCOUNTER\_DATE\_TIME

**FROM** ANIMAL\_FACTS af **JOIN** PET p **ON** af.PETID **=** p.PETID

**JOIN** ENCOUNTER\_HISTORY eh **ON** p.PETID **=** eh.PETID

EXECUTE PROC\_DEATH\_RECORD('Y', 46, '03-JUL-18', 18, 'Patient crashed during cancer operation, unable to recover');'

##### PROC\_DRG\_STOCK

**CREATE OR REPLACE** PROCEDURE PROC\_DRG\_STOCK(

    pv\_drugid1 **IN** int,

    pv\_drug\_units\_dispensed1 **IN** *NUMBER*

    )

**AS**

    lv\_drugonhand PHARMACOLOGY\_STOCK.DRUG\_UNITS\_INV%**TYPE**;

    lv\_order\_level PHARMACOLOGY\_STOCK.ORDER\_LEVEL%**TYPE**;

**BEGIN**

**SELECT** DRUG\_UNITS\_INV, ORDER\_LEVEL

**INTO** lv\_drugonhand, lv\_order\_level

**FROM** PHARMACOLOGY\_STOCK

**WHERE** DRUGID **=** pv\_drugid1;

**IF** (lv\_drugonhand **-** pv\_drug\_units\_dispensed1 **<=** lv\_order\_level)

**THEN**

**UPDATE** PHARMACOLOGY\_STOCK

**SET** REORDER\_FLAG **=** '1',

            DRUG\_UNITS\_INV **=** (lv\_drugonhand **-** pv\_drug\_units\_dispensed1)

**WHERE** DRUGID **=** pv\_drugid1;

**ELSE**

**UPDATE** PHARMACOLOGY\_STOCK

**SET** DRUG\_UNITS\_INV **=** (lv\_drugonhand **-** pv\_drug\_units\_dispensed1)

**WHERE** DRUGID **=** pv\_drugid1;

**END IF**;

    COMMIT;

**END**;

##### PROC\_NONRX\_VETPROC

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

    p\_vetid **IN** int,

    p\_petid **IN** int,

    p\_vet\_procid **IN** int,

    p\_proc\_notes **IN** *clob*)

**AS**

/\*This is for entering simple procedures with out any medicines administered during

--THIS PROCEDURE HAS BEEN DEPRECIATED SINCE THE PROC\_RX\_VETPROC

allows for procedures without medicine to be entered in.

This is kept in only for testing. \*/

**BEGIN**

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

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

    COMMIT;

**END**;

##### PROC\_PATH\_RESULTS

**CREATE OR REPLACE** PROCEDURE PROC\_PATH\_RESULTS(

    p\_laborderid **IN** PATHOLOGY\_LAB\_ORDERS.LABORDERID%**TYPE**,

    p\_critdisease **IN** PATHOLOGY\_HISTORY.CRITICAL\_DISEASE%**TYPE**,

    p\_results **IN** PATHOLOGY\_HISTORY.RESULTS%**TYPE**

    )

**AS**

    lv\_labid PATHOLOGY\_LAB\_ORDERS.LABID%**TYPE**;

    lv\_kits PATHOLOGY\_LAB\_TESTS.KITS\_ON\_HAND%**TYPE**;

    missinglab\_ex **EXCEPTION**;

**BEGIN**

    --begin section to remove one lab kit from inventory

**SELECT** LABID

**INTO** lv\_labid

**FROM** PATHOLOGY\_LAB\_ORDERS

**WHERE** LABORDERID **=** p\_laborderid;

**SELECT** KITS\_ON\_HAND

**INTO** lv\_kits

**FROM** PATHOLOGY\_LAB\_TESTS

**WHERE** LABID **=** lv\_labid;

**UPDATE** PATHOLOGY\_LAB\_TESTS

**SET** KITS\_ON\_HAND **=** lv\_kits **-** 1

**WHERE** LABID **=** lv\_labid;

    --END labkit inventory UPDATE

    --update the completeion date to today's date

**UPDATE** PATHOLOGY\_LAB\_ORDERS

**SET** DATE\_COMPLETED **=** SYSDATE

**WHERE** LABORDERID **=** p\_laborderid;

    COMMIT;

    --begin final merge into the patient chart

**MERGE** **INTO** PATHOLOGY\_HISTORY ph

**USING** PATHOLOGY\_LAB\_ORDERS po

**ON** (ph.LABORDERID **=** po.LABORDERID)

**WHEN** **MATCHED** **THEN**

**UPDATE** **SET** ph.DATE\_COMPLETED **=** po.DATE\_COMPLETED,

            ph.CRITICAL\_DISEASE **=** p\_critdisease,

            ph.RESULTS **=** p\_results

**WHERE** ph.LABORDERID **=** p\_laborderid;

**EXCEPTION**

**WHEN** missinglab\_ex **THEN**

    DBMS\_OUTPUT.*PUT\_LINE*('The LABORDERID does not exist, LABORDERIDs are generated by the lab\_history table; alert the DBA immediately with this exact');

**END**;

##### PROC\_PRESCRIPTION

**create or replace** PROCEDURE PROC\_PRESCRIPTION

(

p\_vetid **IN** int,

p\_petid **IN** int,

p\_drugid **IN** int,

p\_drug\_units\_prescribed **IN** RX\_ORDER.DRUG\_UNITS\_PRESCRIBED%**TYPE**,

/\*p\_drug\_units\_dispensed IN RX\_ORDER.DRUG\_UNITS\_DISPENSED%TYPE,\*/

p\_times\_per\_day **IN** RX\_ORDER.TIMES\_PER\_DAY%**TYPE**,

p\_drug\_dosage **IN** RX\_HISTORY.DRUG\_DOSAGE%**TYPE**,

p\_date\_written **IN** RX\_HISTORY.DATE\_WRITTEN%**TYPE**,

p\_refills **IN** int)

**IS**

lv\_isitcontrolled PHARMACOLOGY\_STOCK.IS\_CONTROLLED%**TYPE**;

ex\_unique\_day **EXCEPTION**;

**PRAGMA EXCEPTION\_INIT** (ex\_unique\_day, **-**1);

ex\_controlled **EXCEPTION**;

**PRAGMA EXCEPTION\_INIT** (ex\_controlled, **-**02290);

lv\_controlcheck *NUMBER*(9, 2);

lv\_rxid int;

**BEGIN**

    --do the math to prepare verification of controlled substance rules

**SELECT** IS\_CONTROLLED

**INTO** lv\_isitcontrolled

**FROM** PHARMACOLOGY\_STOCK

**WHERE** DRUGID **=** p\_drugid;

**CASE**

**WHEN** lv\_isitcontrolled **IN** ('0', 'n', 'N') **THEN**

        lv\_controlcheck **:=** 0; --will inform that zero values mean non controlled meds.

**ELSE**

            lv\_controlcheck **:=** FUNC\_RX\_APPROVAL(p\_drug\_units\_prescribed, p\_times\_per\_day);

**END CASE**;

**INSERT** **INTO** RX\_HISTORY(VETID, PETID, DRUGID, DRUG\_DOSAGE, DRUG\_UNITS\_PRESCRIBED, TIMES\_PER\_DAY, DATE\_WRITTEN)

**VALUES**(p\_vetid, p\_petid, p\_drugid, p\_drug\_dosage, p\_drug\_units\_prescribed, p\_times\_per\_day, p\_date\_written);

    COMMIT;

**MERGE** **INTO** RX\_ORDER ro

**USING** RX\_HISTORY rh

**ON** (ro.RXID **=** rh.RXID **AND**

ro.DATE\_SUBMITTED **=** rh.DATE\_WRITTEN

**AND** ro.DRUGID **=** ro.DRUGID)

/\*WHEN MATCHED THEN

UPDATE

SET ro.RX\_ORDER\_NOTES = 'Order approved on: '|| SYSDATE || ' Begin notes additional notes here:'

WHERE ro.RXID = lv\_rxid

\*/

**WHEN** **NOT** **MATCHED** **THEN**

**INSERT** (RXID, VETID, PETID, DRUGID, DRUG\_UNITS\_PRESCRIBED, TIMES\_PER\_DAY, DRUG\_DOSAGE, DATE\_SUBMITTED, CONTROLLED\_CHECKER, NUM\_REFILLS\_LEFT)

**VALUES**(rh.RXID, rh.VETID, rh.PETID, rh.drugid, rh.DRUG\_UNITS\_PRESCRIBED, rh.TIMES\_PER\_DAY, rh.DRUG\_DOSAGE, rh.DATE\_WRITTEN, lv\_controlcheck, p\_refills);

commit;

**IF** p\_refills **>** 0

**THEN**

**SELECT** RXID

**INTO** lv\_rxid

**FROM** RX\_HISTORY

**WHERE** VETID **=** p\_vetid **AND**

        PETID **=** p\_petid **AND**

        DRUGID **=** p\_drugid **AND**

        DRUG\_DOSAGE **=** p\_drug\_dosage **AND**

        TIMES\_PER\_DAY **=** p\_times\_per\_day **AND**

        DATE\_WRITTEN **=** p\_date\_written;

**INSERT** **INTO** RX\_REFILLS(RXID, NUM\_REFILLS\_LEFT)

**VALUES** (lv\_rxid, p\_refills);

**END IF**;

COMMIT;

**EXCEPTION**

**WHEN** *DUP\_VAL\_ON\_INDEX* **THEN**

**DECLARE**

        lv\_rxid1 int;

        lv\_rx\_histnotes1 *clob*;

        lv\_rx\_ordernotes1 *clob*;

**BEGIN**

**SELECT** RXID

**INTO** lv\_rxid1

**FROM** RX\_HISTORY

**WHERE** VETID **=** p\_vetid **AND**

        PETID **=** p\_petid **AND**

        DRUGID **=** p\_drugid **AND**

        DRUG\_DOSAGE **=** p\_drug\_dosage **AND**

        TIMES\_PER\_DAY **=** p\_times\_per\_day **AND**

        DATE\_WRITTEN **=** p\_date\_written;

**SELECT** rh.NOTES, ro.RX\_ORDER\_NOTES

**INTO** lv\_rx\_histnotes1, lv\_rx\_ordernotes1

**FROM** RX\_HISTORY rh **JOIN** RX\_ORDER ro **USING**(RXID)

**WHERE** RXID **=** lv\_rxid1;

               /\*WHY 2 DIFFERENT VARIABLES FOR SEEMINGLY THE SAME THING?

            IT'S POSSIBLE THE PHARMACIST AND THE VET MAY WANT DIFFERENT TEXT IN THEIR MESSAGES IN THE FUTURE, THIS WILL MAKE THAT EASER

            AND IS COMMONLY CALLED FORESIGHT\*/

       lv\_rx\_ordernotes1 **:=** lv\_rx\_ordernotes1 **||** CHR(10) **||**'Order approved on: ' **||** p\_date\_written **||** CHR(10)**||** 'Attempt at duplicate submission detected & stopped on'**||** SYSTIMESTAMP **||** CHR(10) **||**'Begin notes additional notes here:'**||** CHR(10);

       lv\_rx\_histnotes1 **:=** lv\_rx\_histnotes1 **||** CHR(10) **||**'Order approved on: ' **||** p\_date\_written **||** CHR(10)**||** 'Attempt at duplicate submission detected & stopped on'**||** SYSTIMESTAMP **||** CHR(10) **||**'Begin notes additional notes here:'**||** CHR(10);

**UPDATE** RX\_ORDER

**SET** RX\_ORDER\_NOTES **=** lv\_rx\_ordernotes1

**WHERE** RXID **=** lv\_rxid1;

**UPDATE** RX\_HISTORY

**SET** NOTES **=** lv\_rx\_histnotes1

**WHERE** RXID **=** lv\_rxid1;

        DBMS\_OUTPUT.*PUT\_LINE*('You have attempted to enter a duplicate prescription, please submit again tomorrow, or correct the error');

        COMMIT;

**END**;

**WHEN** ex\_controlled --PUT NOTES INTO THE ATTEMPT AT FILLING AN ILLEGAL RX

**THEN**

**DECLARE**

   lv\_rxid2 int;

   lv\_rx\_histnotes2 *clob*;

**BEGIN**

**SELECT** RXID

**INTO** lv\_rxid2

**FROM** RX\_HISTORY

**WHERE** VETID **=** p\_vetid **AND**

        PETID **=** p\_petid **AND**

        DRUGID **=** p\_drugid **AND**

        DRUG\_DOSAGE **=** p\_drug\_dosage **AND**

        TIMES\_PER\_DAY **=** p\_times\_per\_day **AND**

        DATE\_WRITTEN **=** p\_date\_written;

**SELECT** rh.NOTES

**INTO** lv\_rx\_histnotes2

**FROM** RX\_HISTORY rh

**WHERE** rh.RXID **=** lv\_rxid2;

        lv\_rx\_histnotes2 **:=** lv\_rx\_histnotes2 **||**CHR(10)**||** 'On ' **||** SYSTIMESTAMP**||** ' an attempt to prescribe a controlled medicine longer than legally allowed was made, this order cannot be filled.';

**UPDATE** RX\_HISTORY

**SET** NOTES **=** lv\_rx\_histnotes2

**WHERE** RXID **=** lv\_rxid2;

        DBMS\_OUTPUT.*PUT\_LINE*('ATTEMPT TO FILL PRESCRIPTION FOR TIME PERIOD LONGER THAN LEGALLY ALLOWED');

**END**;

**END**;

##### PROC\_PATHOL\_ORDER

**CREATE OR REPLACE** PROCEDURE PROC\_PATHOL\_ORDER(

    p\_petid **IN** PATHOLOGY\_HISTORY.PETID%**TYPE**,

    p\_vetid **IN** PATHOLOGY\_HISTORY.VETID%**TYPE**,

    p\_labid **IN** PATHOLOGY\_HISTORY.LABID%**TYPE**

    )

/\*USED BY VETS TO PLACE AN ORDER INTO THE PATHOLOGY LAB\*/

**AS**

lv\_laborderid PATHOLOGY\_HISTORY.LABORDERID%**TYPE**;

**BEGIN**

**INSERT** **INTO** PATHOLOGY\_HISTORY(PETID, VETID, LABID)

**VALUES** (p\_petid, p\_vetid, p\_labid)

**RETURNING** LABORDERID **INTO** lv\_laborderid;

    COMMIT;

**INSERT** **INTO** PATHOLOGY\_LAB\_ORDERS(LABORDERID, PETID, VETID, LABID, DATE\_ORDERED)

**VALUES**(lv\_laborderid,p\_petid, p\_vetid, p\_labid, SYSDATE);

    COMMIT;

**END**;

##### PROC\_RX\_FILL

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

    pv\_rxid **IN** int,

    pv\_staffid **IN** int,

    pv\_drug\_units\_dispensed **IN** *NUMBER*,

    pv\_drug\_units\_prescribed **IN** *NUMBER*

    )

**IS**

    lv\_drug\_units\_rx *number* (9,2);

    lv\_drugid int;

    lv\_refills int;

    lv\_drug\_unit\_inv *number* (9, 2);

    lv\_order\_level *number*(7,2);

    lv\_reorder\_flag *char*(1);

    lv\_rx\_refillid int;

    lv\_drugonhand *number* (9,2);

    lv\_num\_refills\_left int;

    ex\_inventory **EXCEPTION**;

**BEGIN**

**SELECT** NUM\_REFILLS\_LEFT, DRUG\_UNITS\_PRESCRIBED, DRUGID

**INTO** lv\_refills, lv\_drug\_units\_rx, lv\_drugid

**FROM** RX\_ORDER

**WHERE** RXID **=** pv\_rxid;

**SELECT** DRUG\_UNITS\_INV

**INTO** lv\_drug\_unit\_inv

**FROM** PHARMACOLOGY\_STOCK

**WHERE** DRUGID **=** lv\_drugid;

**IF** pv\_drug\_units\_prescribed **>** lv\_drug\_unit\_inv

**THEN**

        RAISE ex\_inventory;

**ELSIF** lv\_refills **>** 0

**THEN**

        lv\_refills **:=** lv\_refills **-** 1;

**UPDATE** RX\_ORDER

**SET** NUM\_REFILLS\_LEFT **=** lv\_refills,

        FILLED\_BY **=** pv\_staffid,

        DATE\_FILLED **=** SYSTIMESTAMP,

        DRUG\_UNITS\_DISPENSED **=** pv\_drug\_units\_dispensed

**WHERE** RXID **=** pv\_rxid;

        COMMIT;

        --update the stock table

        PROC\_DRG\_STOCK(lv\_drugid, pv\_drug\_units\_dispensed);

        --SELECTING THE HIGHEST REFILLID ASSOCIATED WITH A RXID ENSURES WE'LL GET THE MOST RECENT UP TO DATE REFILLID

**SELECT** MAX(REFILLID)

**INTO** lv\_rx\_refillid

**FROM** RX\_REFILLS

**WHERE** RXID **=** pv\_rxid

**GROUP BY** rxid;

**UPDATE** RX\_REFILLS

**SET** DATE\_FILLED **=** SYSTIMESTAMP

**WHERE** REFILLID **=** lv\_rx\_refillid;

        /\*VERIFY NUM REFILLS LEFT BEFORE PROCEEDING\*/

**SELECT** NUM\_REFILLS\_LEFT

**INTO** lv\_num\_refills\_left

**FROM** RX\_REFILLS

**WHERE** REFILLID **=** lv\_rx\_refillid;

**IF** lv\_num\_refills\_left **>** 0 **AND** (lv\_num\_refills\_left **-** 1) **=** lv\_refills

**THEN**

**INSERT** **INTO** RX\_REFILLS (RXID, NUM\_REFILLS\_LEFT)

**VALUES**(pv\_rxid, lv\_refills);

            COMMIT;

**ELSE**

            DBMS\_OUTPUT.*PUT\_LINE*('PROBLEM WITH REFILL VARIABLES NOT MATCHING');

**END IF**;

**ELSE**

**UPDATE** RX\_ORDER

**SET** FILLED\_BY **=** pv\_staffid,

        DATE\_FILLED **=** SYSTIMESTAMP,

        DRUG\_UNITS\_DISPENSED **=** pv\_drug\_units\_dispensed

**WHERE** RXID **=** pv\_rxid;

        COMMIT;

        PROC\_DRG\_STOCK(lv\_drugid, pv\_drug\_units\_dispensed);

**END IF**;

**EXCEPTION**

**WHEN** ex\_inventory **THEN**

**DECLARE**

lv\_rx\_notes *clob*;

**BEGIN**

**SELECT** RX\_ORDER\_NOTES

**INTO** lv\_rx\_notes

**FROM** RX\_ORDER

**WHERE** RXID **=** pv\_rxid;

        --add in new notes when not enough stock

        lv\_rx\_notes **:=** lv\_rx\_notes **||** CHR(10) **||**

            'Attempt made to fill RX with less than on on hand inventory, please consult Vet to discuss re-writing RX as a RX with refills'**||**

            CHR(10)**||** 'This RX will not be filled. The time of this notification is: '**||** SYSTIMESTAMP;

**UPDATE** RX\_ORDER

**SET** RX\_ORDER\_NOTES **=** lv\_rx\_notes

**WHERE** RXID **=** pv\_rxid;

        COMMIT;

**END**;

**END**;

##### PROC\_RX\_VETPROC

**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**;

##### PROC\_ESTIMATE\_BUILD

**CREATE OR REPLACE** PROCEDURE PROC\_ESTIMATE\_BUILD(p\_petid **IN** int)

**AS**

/\*THIS PROCEDURE IS STEP ONE IN BUILDING AN ESTIMATE\*/

lv\_addon\_description ESTIMATE.ADD\_ONS\_DESCRIPTION%**TYPE** **:=** NULL;

lv\_addon\_cost ESTIMATE.INDIVIDUAL\_ADD\_ON\_COST%**TYPE** **:=** 0.00;

lv\_estimateid ESTIMATE.EstimateID%**TYPE** **:=** NULL;

**BEGIN**

/\*FIRST we have to determine if an estimate has already been started for the customer\*/

--The OVER PARTITION is a way of getting a single row group by ...sort of.

**SELECT** ESTIMATEID

**INTO** lv\_estimateid

**FROM** (**SELECT** ESTIMATEID, PETID, DATE\_ESTIMATE\_CREATION,

ROW\_NUMBER() **OVER** (**PARTITION BY** ESTIMATEID **ORDER BY** 1) **as** rn

**FROM** ESTIMATE)

**WHERE** rn **=** 1

**AND** PETID **=** p\_petid

**AND** TRUNC(DATE\_ESTIMATE\_CREATION) **=** TRUNC(SYSDATE);

/\*NEXT we have to determine if the ESTIMATEID IS NULL OR NOT\*/

--IF lv\_estimateid IS NOT NULL THEN

DBMS\_OUTPUT.*PUT\_LINE*('An invoice has been generated for this customer already that invoice # is : '**||**lv\_estimateid);

DBMS\_OUTPUT.*PUT\_LINE*('Please use that ID to add onto the invoice, or start a new one tomorrow.');

--END IF;

--EVEN THOUGH THE ERROR IS WHEN AN ESTIMATE EXISTS WE CAN USE AN EXCEPTION TO CREATE A NEW INVOICE AND HAVE THE ERROR BE THE MEAT OF THE PROGRAM

**EXCEPTION**

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

**SELECT** ESTIMATE\_SEQ**.nextval**

**INTO** lv\_estimateid

**FROM** DUAL;

**INSERT** **INTO** ESTIMATE(ESTIMATEID, PETID, ADD\_ONS\_DESCRIPTION, INDIVIDUAL\_ADD\_ON\_COST, DATE\_ESTIMATE\_CREATION)

**VALUES**(lv\_estimateid, p\_petid, 'StartEstimate', 0.00, SYSDATE);

DBMS\_OUTPUT.*PUT\_LINE*('New EstimateID is: '**||** lv\_estimateid);

**END**

##### PROC\_ESTIMATE\_ADD

**CREATE OR REPLACE** PROCEDURE PROC\_ESTIMATE\_ADD(

    p\_estid **IN** ESTIMATE.ESTIMATEID%**TYPE**,

    p\_petid **IN** ESTIMATE.PETID%**TYPE**,

    p\_vetid **IN** ESTIMATE.VETID%**TYPE** **:=** NULL,

    p\_drugid **IN** ESTIMATE.DrugID%**TYPE** **:=** NULL,

    p\_drug\_units\_rxd **IN** ESTIMATE.Drug\_Units\_Prescribed%**TYPE** **:=** NULL,

    p\_labid **IN** ESTIMATE.LabID%**TYPE** **:=** NULL,

    p\_vetprocid **IN** ESTIMATE.Vet\_ProcedureID%**TYPE** **:=** NULL,

    p\_addon\_descript **IN** ESTIMATE.Add\_Ons\_Description%**TYPE** **:=** NULL,

    p\_addon **IN** ESTIMATE.INDIVIDUAL\_Add\_On\_Cost%**TYPE** **:=** NULL

    )

**AS**

/\*THIS PROCEDURE IS STEP TWO IN BUILDING AN ESTIMATE THIS ADDS ONE LINE TO THE ESTIMATE\*/

lv\_drugid ESTIMATE.DrugID%**TYPE** **:=** NULL;

lv\_drug\_cpu ESTIMATE.Drug\_Cost\_Per\_Unit%**TYPE** **:=** NULL;

lv\_is\_surgery VET\_PROCEDURE.IS\_SURGERY%**TYPE** **:=** NULL;

lv\_rx\_cost ESTIMATE.Rx\_Cost%**TYPE** **:=** NULL;

lv\_labid ESTIMATE.LabID%**TYPE** **:=** NULL;

lv\_lab\_cost ESTIMATE.Lab\_Cost%**TYPE** **:=** NULL;

lv\_specialty\_add\_on\_cost ESTIMATE.Specialty\_Add\_On\_Cost%**TYPE** **:=** NULL;

lv\_specialtyid ESTIMATE.SpecialtyID%**TYPE** **:=** NULL;

lv\_proccost ESTIMATE.Procedure\_Cost%**TYPE** **:=** NULL;

lv\_subtotal ESTIMATE.LINE\_SUBTOTAL%**TYPE** **:=** NULL;

lv\_orfee ESTIMATE.OR\_FEE%**TYPE** **:=** NULL;

e\_toomany **EXCEPTION**;

**BEGIN**

**IF** ((p\_drugid IS NOT NULL **AND** (p\_labid IS NOT NULL **OR** p\_vetprocid IS NOT NULL **OR** p\_addon IS NOT NULL)) **OR**

    (p\_labid IS NOT NULL **AND** (p\_drugid IS NOT NULL **OR** p\_vetprocid IS NOT NULL **OR** p\_addon IS NOT NULL))  **OR**

    (p\_vetprocid IS NOT NULL **AND** (p\_labid IS NOT NULL **OR** p\_drugid IS NOT NULL **OR** p\_addon IS NOT NULL)) **OR**

    (p\_addon IS NOT NULL **AND** (p\_labid IS NOT NULL **OR** p\_vetprocid IS NOT NULL **OR** p\_drugid IS NOT NULL)))

**THEN**

        RAISE e\_toomany;

**ELSE**

**CASE** **WHEN** p\_drugid IS NOT NULL **THEN** --BEGIN PRESCRIPTION ESTIMATE

**SELECT** DRUG\_COST\_PER\_UNIT

**INTO** lv\_drug\_cpu

**FROM** PHARMACOLOGY\_STOCK

**WHERE** DRUGID **=** p\_drugid;

            lv\_rx\_cost **:=** ( lv\_drug\_cpu **\*** p\_drug\_units\_rxd);

            lv\_subtotal **:=** lv\_rx\_cost; --FOR CONSISTENCY AND TO MAKE SUMMING EASIER

**INSERT** **INTO** ESTIMATE (ESTIMATEID, PETID, VETID, DATE\_ESTIMATE\_CREATION, DRUGID, DRUG\_COST\_PER\_UNIT, DRUG\_UNITS\_PRESCRIBED, RX\_COST, LINE\_SUBTOTAL)

**VALUES**(p\_estid, p\_petid, p\_vetid, SYSDATE, p\_drugid, lv\_drug\_cpu, p\_drug\_units\_rxd, lv\_rx\_cost, lv\_subtotal);

            DBMS\_OUTPUT.*PUT\_LINE*('VALUES ENTERED IN TO NEW ESTIMATE LINE: '**||**

                            CHR(34)**||**'|'**||**p\_estid**||**'|'**||**p\_petid**||**'|'**||**p\_vetid**||**'|'**||**SYSDATE**||**'|'

**||**p\_drugid**||**'|'**||**lv\_drug\_cpu**||**'|'**||**p\_drug\_units\_rxd**||**'|'**||**lv\_rx\_cost**||**'|'**||**lv\_subtotal**||**'| END');

            --END PRESCRIPTION ESITIMATE

**WHEN** p\_labid IS NOT NULL **THEN** -- BEGIN LAB COST ESTIMATE

**SELECT** LAB\_COST

**INTO** lv\_lab\_cost

**FROM** PATHOLOGY\_LAB\_TESTS

**WHERE** LABID **=** p\_labid;

            lv\_subtotal **:=** lv\_lab\_cost; --FOR CONSISTENCY, AND TO MAKE SUMMING EASIER

**INSERT** **INTO** ESTIMATE (ESTIMATEID, PETID, VETID, DATE\_ESTIMATE\_CREATION, LABID, LAB\_COST, LINE\_SUBTOTAL)

**VALUES** (p\_estid, p\_petid, p\_vetid, SYSDATE, p\_labid, lv\_lab\_cost, lv\_subtotal);

            DBMS\_OUTPUT.*PUT\_LINE*('VALUES ENTERED IN TO NEW ESTIMATE LINE: '**||**

                        CHR(34)**||**'|'**||**p\_estid**||**'|'**||**p\_petid**||**'|'**||**p\_vetid**||**'|'**||**SYSDATE**||**'|'

**||**p\_labid**||**'|'**||**lv\_lab\_cost**||**'|'**||**lv\_subtotal**||**'|'**||**'| END');

        --END LAB ESTIMATE

**WHEN** p\_vetprocid IS NOT NULL **THEN** --BEGIN PROCEDURE COST CALULATION

        --GET INFORMATION ABOUT THE PROCEDURE INCLUDING IF THE OPERATING THEATER WAS USED

**SELECT** SPECIALTYID, IS\_SURGERY, VET\_PROCEDURE\_COST

**INTO** lv\_specialtyid, lv\_is\_surgery, lv\_proccost

**FROM** VET\_PROCEDURE

**WHERE** VET\_PROCEDUREID **=** p\_vetprocid;

        --GET THE COST OF THE SPECIALTY ADD ON VALUE FOR A SPECIALIZED APPOINTMENT (EXAMPLE: VETERINARY ONCOLOGY)

**IF** lv\_specialtyid IS NOT NULL **THEN**

**SELECT** SPECIALTY\_ADD\_ON\_COST

**INTO** lv\_specialty\_add\_on\_cost

**FROM** SPECIALTIES

**WHERE** SPECIALTYID **=** lv\_specialtyid;

**ELSE** lv\_specialty\_add\_on\_cost **:=** 0.00;

**END IF**;

        --IF OPERATING THEATER WAS USED ADD THE COST

**IF** lv\_is\_surgery **IN** ('1', 'y', 'Y', 't', 'T') **THEN**

                lv\_orfee **:=** 250.00; --OPERATING THEATER OR\_FEE

**ELSE** lv\_orfee **:=** 0.00;

**END IF**;

            lv\_subtotal **:=** lv\_proccost **+** lv\_specialty\_add\_on\_cost **+** lv\_orfee;

**INSERT** **INTO** ESTIMATE (ESTIMATEID, PETID, VETID, DATE\_ESTIMATE\_CREATION, SPECIALTY\_ADD\_ON\_COST, SPECIALTYID, VET\_PROCEDUREID, PROCEDURE\_COST, OR\_FEE, LINE\_SUBTOTAL)

**VALUES** (p\_estid, p\_petid, p\_vetid, SYSDATE, lv\_specialty\_add\_on\_cost, lv\_specialtyid, p\_vetprocid, lv\_proccost, lv\_orfee, lv\_subtotal);

            DBMS\_OUTPUT.*PUT\_LINE*('VALUES ENTERED IN TO NEW ESTIMATE LINE: '**||**

                    CHR(34)**||**'|'**||**p\_estid**||**'|'**||**p\_petid**||**'|'**||**p\_vetid**||**'|'**||**SYSDATE**||**'|'**||**lv\_specialty\_add\_on\_cost**||**'|'**||**lv\_specialtyid**||**'|'

**||**p\_vetprocid**||**'|'**||**lv\_proccost**||**'|'**||**lv\_orfee**||**'|'**||**lv\_subtotal**||**'| END');

        --END PROCEDURE CALULATION

**WHEN** p\_addon IS NOT NULL **THEN** --BEGIN ADD ON COSTS (LIKE MERCHANDISE, AND TREAT PURCHASES FROM THE FRONT OFFICE)

        --NO NEED FOR SELECTS

        --TOTAL\_ADD\_ON\_COSTS WILL BE NULL FOR NOW

            lv\_subtotal **:=** p\_addon;

**INSERT** **INTO** ESTIMATE (ESTIMATEID, PETID, VETID, DATE\_ESTIMATE\_CREATION, INDIVIDUAL\_ADD\_ON\_COST, ADD\_ONS\_DESCRIPTION, TOTAL\_ADD\_ON\_COSTS, LINE\_SUBTOTAL)

**VALUES** (p\_estid, p\_petid, p\_vetid, SYSDATE, p\_addon, p\_addon\_descript, NULL, lv\_subtotal);

            DBMS\_OUTPUT.*PUT\_LINE*('VALUES ENTERED IN TO NEW ESTIMATE LINE: '**||**

                                CHR(34)**||**'|'**||**p\_estid**||**'|'**||**p\_petid**||**'|'**||**p\_vetid**||**'|'**||**SYSDATE**||**'|'**||**p\_addon**||**'|'**||**

                                p\_addon\_descript**||**'|'**||**NULL**||**'|'**||**lv\_subtotal**||**'| END');

**ELSE** DBMS\_OUTPUT.*PUT\_LINE*('You have entered in no useful data, no insert has been done; try again!');

**END CASE**;

**END IF**;

**EXCEPTION**

**WHEN** e\_toomany **THEN**

    DBMS\_OUTPUT.*PUT\_LINE*('You may only enter one type of data per line' **||** CHR(34)**||** 'One procedure, or one operation, or one bag of treats etc.');

**END**;

EXECUTE proc\_estimate\_add(P\_ESTID **=>** 100, P\_PETID **=>** 4, P\_DRUGID **=>** 3, P\_DRUG\_UNITS\_RXD **=>** 1);

EXECUTE proc\_estimate\_add(100, 4, NULL, NULL, NULL, NULL, 10);

EXECUTE proc\_estimate\_add(100, 4, P\_VETPROCID **=>** 16);

EXECUTE proc\_estimate\_add(101, 23, NULL, 9, 2);

EXECUTE proc\_estimate\_add(101, 23, P\_VETPROCID **=>** 11);

EXECUTE proc\_estimate\_add(101, 23, NULL, NULL, NULL, 22);

EXECUTE proc\_estimate\_add(101, 23, NULL, NULL, NULL, NULL, NULL, 'Kitty Treats', 9.99);

##### PROC\_EXPIRE\_EST

**CREATE OR REPLACE** PROCEDURE PROC\_EXPIRE\_EST

**AS**

**BEGIN**

**DELETE** **FROM** ESTIMATE

**WHERE** TRUNC(DATE\_ESTIMATE\_CREATION) **<=** TRUNC(SYSDATE) **-** 30;

COMMIT;

**EXCEPTION**

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

    DBMS\_OUTPUT.*PUT\_LINE*('No expired estimates today!');

**END**;

##### PROC\_GETINVNUM

**CREATE OR REPLACE** PROCEDURE PROC\_GETINVNUM(p\_petid **IN** int, p\_invid **OUT** int)

**AS**

**BEGIN**

**SELECT** **\***

**INTO** p\_invid

**FROM** (**SELECT** INVOICEID

**FROM** INVOICE\_OPEN

**WHERE** trunc(DATE\_INVOICE\_CREATION) **=** TRUNC(SYSDATE) **AND** PETID **=** p\_petid

**ORDER BY** LINEID **DESC**)

**WHERE** **ROWNUM=**1;

DBMS\_OUTPUT.*PUT\_LINE*(p\_invid);

**EXCEPTION** **WHEN** *NO\_DATA\_FOUND* **THEN** --WHEN NO DATA FOUND GENERATE AN INVOICE ID

**SELECT** INVOICE\_SEQ**.nextval**

**INTO** p\_invid

**FROM** DUAL;

**INSERT** **INTO** INVOICE\_OPEN(INVOICEID, PETID, DATE\_INVOICE\_CREATION, ADD\_ONS\_DESCRIPTION, INDIVIDUAL\_ADD\_ON\_COST, LINE\_SUBTOTAL)

**VALUES**(p\_invid, p\_petid, SYSDATE, 'New Invoice Start',0.00, NULL );

COMMIT;

**END**;

##### PROC\_ADD\_TO\_INVOICE

**CREATE OR REPLACE** PROCEDURE PROC\_ADD\_TO\_INVOICE(

p\_invoiceid **IN** INVOICE\_OPEN.INVOICEID%**TYPE**,

p\_petid **IN** INVOICE\_OPEN.PETID%**TYPE**,

p\_vetid **IN** INVOICE\_OPEN.VETID%**TYPE** **DEFAULT** NULL,

p\_date\_invoice\_creation **IN** INVOICE\_OPEN.DATE\_INVOICE\_CREATION%**TYPE** **DEFAULT** NULL,

p\_rxid **IN** INVOICE\_OPEN.RXID%**TYPE** **DEFAULT** NULL,

p\_drugid **IN** INVOICE\_OPEN.DRUGID%**TYPE** **DEFAULT** NULL,

p\_drug\_cost\_per\_unit **IN** INVOICE\_OPEN.DRUG\_COST\_PER\_UNIT%**TYPE** **DEFAULT** NULL,

p\_drug\_units\_dispensed **IN** INVOICE\_OPEN.DRUG\_UNITS\_DISPENSED%**TYPE** **DEFAULT** NULL,

p\_labid **IN** INVOICE\_OPEN.LABID%**TYPE** **DEFAULT** NULL,

p\_lab\_cost **IN** INVOICE\_OPEN.LAB\_COST%**TYPE** **DEFAULT** NULL,

p\_specialty\_add\_on\_cost **IN** INVOICE\_OPEN.SPECIALTY\_ADD\_ON\_COST%**TYPE** **DEFAULT** NULL,

p\_specialtyid **IN** INVOICE\_OPEN.SPECIALTYID%**TYPE** **DEFAULT** NULL,

p\_vet\_procedureid **IN** INVOICE\_OPEN.VET\_PROCEDUREID%**TYPE** **DEFAULT** NULL,

p\_or\_fee **IN** INVOICE\_OPEN.OR\_FEE%**TYPE** **DEFAULT** NULL,

p\_proc\_cost **IN** INVOICE\_OPEN.PROCEDURE\_COST%**TYPE** **DEFAULT** NULL,

p\_add\_ons\_descrip **IN** INVOICE\_OPEN.ADD\_ONS\_DESCRIPTION%**TYPE** **DEFAULT** NULL,

p\_indv\_add\_on\_cost **IN** INVOICE\_OPEN.INDIVIDUAL\_ADD\_ON\_COST%**TYPE** **DEFAULT** NULL

)

**AS**

--calculates LINE\_SUBTOTAL in this procedure for safety

lv\_line\_subtotal INVOICE\_OPEN.LINE\_SUBTOTAL%**TYPE** **:=** NULL;

lv\_rx\_cost INVOICE\_OPEN.RX\_COST%**TYPE** **:=** NULL;

e\_nocosts **EXCEPTION**;

**BEGIN**

DBMS\_OUTPUT.*PUT\_LINE*('begin variable display');

DBMS\_OUTPUT.*PUT\_LINE*('p\_lab\_cost: '**||** p\_lab\_cost);

DBMS\_OUTPUT.*PUT\_LINE*('p\_labid: '**||**p\_labid);

**IF** p\_rxid IS NOT NULL **THEN**

        lv\_rx\_cost **:=** (p\_drug\_cost\_per\_unit **\*** p\_drug\_units\_dispensed);

        lv\_line\_subtotal **:=** lv\_rx\_cost; --for consistency

**ELSIF** p\_labid IS NOT NULL **THEN**

        DBMS\_OUTPUT.*PUT\_LINE*('p\_lab\_cost: '**||** p\_lab\_cost);

        lv\_line\_subtotal **:=** p\_lab\_cost; --for consistency

        DBMS\_OUTPUT.*PUT\_LINE*('lv\_line\_subtotal: ' **||**lv\_line\_subtotal);

**ELSIF** p\_vet\_procedureid IS NOT NULL **THEN**

**CASE**

**WHEN** p\_specialty\_add\_on\_cost IS NOT NULL **THEN**

**IF** p\_or\_fee IS NOT NULL **THEN**

                lv\_line\_subtotal **:=** p\_specialty\_add\_on\_cost **+** p\_or\_fee **+** p\_proc\_cost;

**ELSE** lv\_line\_subtotal **:=** p\_specialty\_add\_on\_cost **+** p\_proc\_cost;

**END IF**;

**WHEN** p\_specialty\_add\_on\_cost IS NULL **AND** p\_or\_fee IS NOT NULL **THEN**

            lv\_line\_subtotal **:=** p\_or\_fee **+** p\_proc\_cost;

**ELSE** lv\_line\_subtotal **:=** p\_proc\_cost; --for consistency and to close out all possible permutations of this contingency

**END CASE**;

**ELSIF** p\_indv\_add\_on\_cost IS NOT NULL **THEN**

        lv\_line\_subtotal **:=** p\_indv\_add\_on\_cost;

**ELSE** RAISE e\_nocosts;

**END IF**;

**INSERT** **INTO** INVOICE\_OPEN (INVOICEID, PETID, VETID, DATE\_INVOICE\_CREATION, RXID, DRUGID, DRUG\_COST\_PER\_UNIT, DRUG\_UNITS\_DISPENSED, RX\_COST, LABID, LAB\_COST, SPECIALTY\_ADD\_ON\_COST, SPECIALTYID, VET\_PROCEDUREID, OR\_FEE, PROCEDURE\_COST, ADD\_ONS\_DESCRIPTION, INDIVIDUAL\_ADD\_ON\_COST, LINE\_SUBTOTAL)

**VALUES**(p\_invoiceid, p\_petid, p\_vetid, p\_date\_invoice\_creation, p\_rxid, p\_drugid, p\_drug\_cost\_per\_unit, p\_drug\_units\_dispensed,lv\_rx\_cost, p\_labid, p\_lab\_cost, p\_specialty\_add\_on\_cost, p\_specialtyid, p\_vet\_procedureid, p\_or\_fee, p\_proc\_cost, p\_add\_ons\_descrip, p\_indv\_add\_on\_cost, lv\_line\_subtotal);

    /\*NOTE WE ARE MAKING THE PROGRAMS THAT CALL THIS SUBROUTINE PASS SYSDATE SO THAT WAY WE CAN USE THIS PROGRAM TO ADD UNINTENTIONALLY MISSED LINES TO INVOICES ON

    A CASE BY CASE BASIS \*/

COMMIT;

**EXCEPTION**

**WHEN** e\_nocosts **THEN**

        DBMS\_OUTPUT.*PUT\_LINE*('Nothing that has a cost was entered; even free procedures have a 0.00 cost, so something has gone very wrong, call the DBA');

**END**;

##### PROC\_LATE\_FEE

**CREATE OR REPLACE** PROC\_LATE\_FEE

**AS**

    lv\_percentfee *number*(4,2) **:=** 0.05;

    lv\_feetotal *number*(9,2) **:=** NULL;

    lv\_newtotal *number*(9,2) **:=** NULL;

--DOING THIS WITH A CURSOR, I DON'T TRUST GIANT UPDATE TABLE STATEMENTS BASED ON COMPARISON OPERATIONS

**CURSOR** cur\_lates **IS**

**SELECT** LINEID, LINE\_SUBTOTAL, LATE\_FEE, GRAND\_TOTAL

**FROM** INVOICE\_OPEN

**WHERE** TRUNC(DATE\_INVOICE\_CREATION) **<=** TRUNC(SYSDATE) **-** 30;

**TYPE** type\_late **IS** **RECORD**(

    id INVOICE\_OPEN.LINEID%**TYPE**,

    sub INVOICE\_OPEN.LINE\_SUBTOTAL%**TYPE**,

    latefee INVOICE\_OPEN.LATE\_FEE%**TYPE**,

    grand INVOICE\_OPEN.GRAND\_TOTAL%**TYPE**

    );

rec\_latefee type\_late;

**BEGIN**

**OPEN** cur\_lates;

**LOOP**

**FETCH** cur\_lates **INTO** rec\_latefee;

    lv\_feetotal **:=** (rec\_latefee.sub **\*** lv\_percentfee);

    lv\_newtotal **:=** (rec\_latefee.sub **+** lv\_feetotal);

**UPDATE** INVOICE\_OPEN

**SET** LATE\_FEE **=** lv\_feetotal,

         GRAND\_TOTAL **=** lv\_newtotal

**WHERE** LINEID **=** rec\_latefee.id;

**EXIT** **WHEN** cur\_lates%NOTFOUND;

**END LOOP**;

**EXCEPTION**

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

    DBMS\_OUTPUT.*PUT\_LINE*('No late invoices needed to be updated today!');

**END**;

###### UPDATED PROCEDURES FOR INVOICING

BULK\_RX\_PROC\_APPROVAL (INVOICE SUPPORTED)

**CREATE OR REPLACE** PROCEDURE PROC\_BULK\_RXPROC\_APPROVAL (

    /\*upated for invoicing support\*/

    p\_assent **IN** *char*,

    p\_approver **IN** int)

**AS**

**CURSOR** cur\_procrxs **IS**

**SELECT** PETID, VETID, DRUG\_UNITS\_PRESCRIBED, DRUG\_UNITS\_DISPENSED, DRUGID, RXID

**FROM** DAILY\_PROCRX\_V;

**TYPE** type\_rxupdate **IS** **RECORD**

    (

        pet DAILY\_PROCRX\_V.PETID%**TYPE**,

        vet DAILY\_PROCRX\_V.VETID%**TYPE**,

        drugrxd DAILY\_PROCRX\_V.DRUG\_UNITS\_PRESCRIBED%**TYPE**,

        drugdisp DAILY\_PROCRX\_V.DRUG\_UNITS\_DISPENSED%**TYPE**,

        drugid RX\_ORDER.DRUGID%**TYPE**,

        rxid1 RX\_ORDER.RXID%**TYPE**

    );

    rec\_rxupdate type\_rxupdate;

    --lv\_petid DAILY\_PROCRX\_V.PETID%TYPE;

    lv\_invnum INVOICE\_OPEN.INVOICEID%**TYPE**;

    lv\_drugcost PHARMACOLOGY\_STOCK.DRUG\_COST\_PER\_UNIT%**TYPE**;

**BEGIN**

**IF** p\_assent **in** ('y','Y','1','t','T')

**THEN**

**OPEN** cur\_procrxs;

**LOOP**

**FETCH** cur\_procrxs **INTO** rec\_rxupdate;

          --lv\_petid := NULL; --reset the PETID each pass through the loop

**UPDATE** RX\_ORDER

**SET** DRUG\_UNITS\_DISPENSED **=** rec\_rxupdate.drugrxd, --they are affirming the amount dispensed are the same prescribed

           FILLED\_BY **=** p\_approver

           --DATE\_FILLED = SYSTIMESTAMP, nope....put this in for when it originally gets put into the system

**WHERE** RXID **=** rec\_rxupdate.rxid1;

DBMS\_OUTPUT.*PUT\_LINE*(rec\_rxupdate.drugid **||**' ' **||**rec\_rxupdate.drugrxd **||**' ' **||** rec\_rxupdate.drugdisp**||**' ' **||** rec\_rxupdate.rxid1**||**' ' **||**rec\_rxupdate.pet**||**' ' **||**rec\_rxupdate.vet);

DBMS\_OUTPUT.*PUT\_LINE*('rec\_rxupdate.drugid' **||**' ' **||**'rec\_rxupdate.drugrxd' **||**' ' **||** 'rec\_rxupdate.drugdisp'**||**' ' **||** 'rec\_rxupdate.rxid1'**||**' ' **||**'rec\_rxupdate.pet'**||**' ' **||**'rec\_rxupdate.vet');

      --  lv\_petid := rec\_rxupdate.pet;

**UPDATE** RX\_HISTORY

**SET** DATE\_FILLED **=** SYSDATE

**WHERE** RXID **=** rec\_rxupdate.rxid1;

          PROC\_DRG\_STOCK(rec\_rxupdate.drugid, rec\_rxupdate.drugrxd); --again by using this program they are affirming what was RX'd is what was filled.

          --check for invoice number

          PROC\_GETINVNUM(rec\_rxupdate.pet, lv\_invnum);

          --get drug cost

**SELECT** DRUG\_COST\_PER\_UNIT

**INTO** lv\_drugcost

**FROM** PHARMACOLOGY\_STOCK

**WHERE** DRUGID **=** rec\_rxupdate.drugid;

          PROC\_ADD\_TO\_INVOICE(p\_invoiceid **=>** lv\_invnum, p\_petid **=>** rec\_rxupdate.pet, p\_vetid **=>** rec\_rxupdate.vet, p\_date\_invoice\_creation **=>** SYSDATE, p\_rxid **=>** rec\_rxupdate.rxid1,

              p\_drugid **=>** rec\_rxupdate.drugid, p\_drug\_cost\_per\_unit **=>** lv\_drugcost, p\_drug\_units\_dispensed **=>** rec\_rxupdate.drugrxd);

           --note for bulk approvals which are done only for procedure dispensed meds the dispensed # is always the same as the prescribed

**EXIT** **WHEN** cur\_procrxs%NOTFOUND;

**END LOOP**;

**END IF**;

**END**;

DBMS\_OUTPUT.*PUT\_LINE*(rec\_rxupdate.drugid **||**' ' **||**rec\_rxupdate.drugrxd **||**' ' **||** rec\_rxupdate.drugdisp**||**' ' **||** rec\_rxupdate.rxid1**||**' ' **||**rec\_rxupdate.pet**||**' ' **||**rec\_rxupdate.vet);

DBMS\_OUTPUT.*PUT\_LINE*('rec\_rxupdate.drugid' **||**' ' **||**'rec\_rxupdate.drugrxd' **||**' ' **||** 'rec\_rxupdate.drugdisp'**||**' ' **||** 'rec\_rxupdate.rxid1'**||**' ' **||**'rec\_rxupdate.pet'**||**' ' **||**'rec\_rxupdate.vet');

EXECUTE PROC\_BULK\_RXPROC\_APPROVAL('y', 1);

PROC\_RX\_FILL (INVOICE SUPPORTED)

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

    /\*upated for invoicing support\*/

    pv\_rxid **IN** int,

    pv\_staffid **IN** int,

    pv\_drug\_units\_dispensed **IN** *NUMBER*,

    pv\_drug\_units\_prescribed **IN** *NUMBER*

    )

**IS**

    lv\_petid RX\_ORDER.PETID%**TYPE**;

    lv\_drug\_units\_rx *number* (9,2);

    lv\_drugid int;

    lv\_refills int;

    lv\_drug\_unit\_inv *number* (9, 2);

    lv\_order\_level *number*(7,2);

    lv\_reorder\_flag *char*(1);

    lv\_rx\_refillid int;

    lv\_drugonhand *number* (9,2);

    lv\_num\_refills\_left int;

    lv\_vetid RX\_ORDER.VETID%**TYPE**;

    lv\_drugcost PHARMACOLOGY\_STOCK.DRUG\_COST\_PER\_UNIT%**TYPE**;

    lv\_linesubtotal INVOICE\_OPEN.LINE\_SUBTOTAL%**TYPE**;

    lv\_invoice\_num INVOICE\_OPEN.INVOICEID%**TYPE**;

    lv\_rxcost INVOICE\_OPEN.RX\_COST%**TYPE**;

    ex\_inventory **EXCEPTION**;

**BEGIN**

**SELECT** NUM\_REFILLS\_LEFT, DRUG\_UNITS\_PRESCRIBED, DRUGID, PETID, VETID

**INTO** lv\_refills, lv\_drug\_units\_rx, lv\_drugid, lv\_petid, lv\_vetid

**FROM** RX\_ORDER

**WHERE** RXID **=** pv\_rxid;

**SELECT** DRUG\_UNITS\_INV, DRUG\_COST\_PER\_UNIT

**INTO** lv\_drug\_unit\_inv, lv\_drugcost

**FROM** PHARMACOLOGY\_STOCK

**WHERE** DRUGID **=** lv\_drugid;

**IF** pv\_drug\_units\_prescribed **>** lv\_drug\_unit\_inv

**THEN**

        RAISE ex\_inventory;

**ELSIF** lv\_refills **>** 0

**THEN**

        lv\_refills **:=** lv\_refills **-** 1;

**UPDATE** RX\_ORDER

**SET** NUM\_REFILLS\_LEFT **=** lv\_refills,

        FILLED\_BY **=** pv\_staffid,

        DATE\_FILLED **=** SYSTIMESTAMP,

        DRUG\_UNITS\_DISPENSED **=** pv\_drug\_units\_dispensed

**WHERE** RXID **=** pv\_rxid;

        COMMIT;

        --update the stock table

        PROC\_DRG\_STOCK(lv\_drugid, pv\_drug\_units\_dispensed);

        --SELECTING THE HIGHEST REFILLID ASSOCIATED WITH A RXID ENSURES WE'LL GET THE MOST RECENT UP TO DATE REFILLID

**SELECT** MAX(REFILLID)

**INTO** lv\_rx\_refillid

**FROM** RX\_REFILLS

**WHERE** RXID **=** pv\_rxid

**GROUP BY** rxid;

**UPDATE** RX\_REFILLS

**SET** DATE\_FILLED **=** SYSTIMESTAMP

**WHERE** REFILLID **=** lv\_rx\_refillid;

        /\*VERIFY NUM REFILLS LEFT BEFORE PROCEEDING\*/

**SELECT** NUM\_REFILLS\_LEFT

**INTO** lv\_num\_refills\_left

**FROM** RX\_REFILLS

**WHERE** REFILLID **=** lv\_rx\_refillid;

**IF** lv\_num\_refills\_left **>** 0 **AND** (lv\_num\_refills\_left **-** 1) **=** lv\_refills

**THEN**

**INSERT** **INTO** RX\_REFILLS (RXID, NUM\_REFILLS\_LEFT)

**VALUES**(pv\_rxid, lv\_refills);

            COMMIT;

**ELSE**

            DBMS\_OUTPUT.*PUT\_LINE*('PROBLEM WITH REFILL VARIABLES NOT MATCHING');

**END IF**;

**ELSE**

**UPDATE** RX\_ORDER

**SET** FILLED\_BY **=** pv\_staffid,

        DATE\_FILLED **=** SYSTIMESTAMP,

        DRUG\_UNITS\_DISPENSED **=** pv\_drug\_units\_dispensed

**WHERE** RXID **=** pv\_rxid;

        COMMIT;

        PROC\_DRG\_STOCK(lv\_drugid, pv\_drug\_units\_dispensed);

**END IF**;

    PROC\_GETINVNUM(lv\_petid, lv\_invoice\_num);

    --p\_invoiceid   p\_petid p\_vetid p\_date\_invoice\_creation p\_rxid  p\_drugid    p\_drug\_cost\_per\_unit    p\_drug\_units\_dispensed

--add to the invoice

    PROC\_ADD\_TO\_INVOICE(p\_invoiceid **=>** lv\_invoice\_num, p\_petid **=>** lv\_petid, p\_vetid **=>** lv\_vetid, p\_date\_invoice\_creation **=>** SYSDATE, p\_rxid **=>** pv\_rxid, p\_drugid **=>** lv\_drugid, p\_drug\_cost\_per\_unit **=>** lv\_drugcost, p\_drug\_units\_dispensed **=>** pv\_drug\_units\_dispensed);

COMMIT;

**EXCEPTION**

**WHEN** ex\_inventory **THEN**

**DECLARE**

lv\_rx\_notes *clob*;

**BEGIN**

**SELECT** RX\_ORDER\_NOTES

**INTO** lv\_rx\_notes

**FROM** RX\_ORDER

**WHERE** RXID **=** pv\_rxid;

        --add in new notes when not enough stock

        lv\_rx\_notes **:=** lv\_rx\_notes **||** CHR(10) **||**

            'Attempt made to fill RX with less than on on hand inventory, please consult Vet to discuss re-writing RX as a RX with refills'**||**

            CHR(10)**||** 'This RX will not be filled. The time of this notification is: '**||** SYSTIMESTAMP;

**UPDATE** RX\_ORDER

**SET** RX\_ORDER\_NOTES **=** lv\_rx\_notes

**WHERE** RXID **=** pv\_rxid;

        COMMIT;

**END**;

**END**;

PROC\_PATH\_RESULTS (INVOICE SUPPORTED)

**CREATE OR REPLACE** PROCEDURE PROC\_PATH\_RESULTS(

    /\*upated for invoicing support\*/

    p\_laborderid **IN** PATHOLOGY\_LAB\_ORDERS.LABORDERID%**TYPE**,

    p\_critdisease **IN** PATHOLOGY\_HISTORY.CRITICAL\_DISEASE%**TYPE**,

    p\_results **IN** PATHOLOGY\_HISTORY.RESULTS%**TYPE**

    )

**AS**

    lv\_labid PATHOLOGY\_LAB\_ORDERS.LABID%**TYPE**;

    lv\_kits PATHOLOGY\_LAB\_TESTS.KITS\_ON\_HAND%**TYPE**;

    lv\_invid INVOICE\_OPEN.INVOICEID%**TYPE**;

    lv\_labcost PATHOLOGY\_LAB\_TESTS.LAB\_COST%**TYPE**;

    lv\_petid ANIMAL\_FACTS.PETID%**TYPE**;

    missinglab\_ex **EXCEPTION**;

**BEGIN**

    --begin section to remove one lab kit from inventory

**SELECT** LABID

**INTO** lv\_labid

**FROM** PATHOLOGY\_LAB\_ORDERS

**WHERE** LABORDERID **=** p\_laborderid;

**SELECT** KITS\_ON\_HAND, LAB\_COST

**INTO** lv\_kits, lv\_labcost

**FROM** PATHOLOGY\_LAB\_TESTS

**WHERE** LABID **=** lv\_labid;

**UPDATE** PATHOLOGY\_LAB\_TESTS

**SET** KITS\_ON\_HAND **=** lv\_kits **-** 1

**WHERE** LABID **=** lv\_labid;

    --END labkit inventory UPDATE

    --update the completeion date to today's date

**UPDATE** PATHOLOGY\_LAB\_ORDERS

**SET** DATE\_COMPLETED **=** SYSDATE

**WHERE** LABORDERID **=** p\_laborderid;

    COMMIT;

    --begin final merge into the patient chart

**MERGE** **INTO** PATHOLOGY\_HISTORY ph

**USING** PATHOLOGY\_LAB\_ORDERS po

**ON** (ph.LABORDERID **=** po.LABORDERID)

**WHEN** **MATCHED** **THEN**

**UPDATE** **SET** ph.DATE\_COMPLETED **=** po.DATE\_COMPLETED,

            ph.CRITICAL\_DISEASE **=** p\_critdisease,

            ph.RESULTS **=** p\_results

**WHERE** ph.LABORDERID **=** p\_laborderid;

--BEGIN INVOICE SECTION

--start by grabbing the PETID

**SELECT** PETID

**INTO** lv\_petid

**FROM** PATHOLOGY\_HISTORY

**WHERE** LABORDERID **=** p\_laborderid ;

    --get or generate invoice number

    PROC\_GETINVNUM(lv\_petid, lv\_invid);

**INSERT** **INTO** INVOICE\_OPEN(INVOICEID, PETID, DATE\_INVOICE\_CREATION, LABID, LAB\_COST)

**VALUES**(lv\_invid, lv\_petid, SYSDATE, lv\_labid, lv\_labcost);

**EXCEPTION**

**WHEN** missinglab\_ex **THEN**

    DBMS\_OUTPUT.*PUT\_LINE*('The LABORDERID does not exist, LABORDERIDs are generated by the lab\_history table; alert the DBA immediately with this exact');

**END**;

PROC\_RX\_VETPROC (INVOICE SUPPORTED)

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

    /\*upated for invoicing support\*/

    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\_invid INVOICE\_OPEN.INVOICEID%**TYPE** **:=** NULL;

lv\_ppprocid int **:=** NULL;

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

lv\_proccost VET\_PROCEDURE.VET\_PROCEDURE\_COST%**TYPE** **:=** NULL;

/\*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**;

/\*Procedures are entered into the system automatically are added to the invoices but the medicines are not until they are verified as removed from inventory

thus those will be added to the invoice upon running the approval script \*/

--GET THE INVOICE NUMBER

    PROC\_GETINVNUM(p\_petid, lv\_invid);

--GET THE COST

**SELECT** VET\_PROCEDURE\_COST

**INTO** lv\_proccost

**FROM** VET\_PROCEDURE

**WHERE** VET\_PROCEDUREID **=** p\_vet\_procid;

--ADD TO INVOICE

    PROC\_ADD\_TO\_INVOICE(p\_invoiceid **=>** lv\_invid, p\_petid **=>** p\_petid, p\_vetid **=>** p\_vetid, p\_date\_invoice\_creation **=>** SYSDATE, p\_vet\_procedureid **=>** p\_vet\_procid, p\_proc\_cost **=>** lv\_proccost);

**END**;

#### Packages

##### CHART\_PKG

--PACKAGE HEADER START

**CREATE OR REPLACE** **PACKAGE** CHART\_PKG

**IS**

pv\_petid ANIMAL\_FACTS.PETID%**TYPE**;

**CREATE OR REPLACE** FUNCTION FUNC\_CHART\_NAME (f\_petid **IN** int)

**RETURN** *varchar2*;

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

    f\_petid **IN** ANIMAL\_FACTS.PETID%**TYPE**)

**RETURN** *clob*;

**CREATE OR REPLACE** FUNCTION FUNC\_FULLCHARTNOTES(f\_petid **IN** ANIMAL\_FACTS.PETID%**TYPE**)

**RETURN** *clob*;

**CREATE OR REPLACE** FUNCTION FUNC\_RX\_CHART\_DETAILS(f\_petid **IN** RX\_HISTORY.PETID%**TYPE**)

**RETURN** *clob*;

**CREATE OR REPLACE** FUNCTION FUNC\_FLTXTCHART (f\_petid **IN** ANIMAL\_FACTS.PETID%**TYPE**)

**return** *clob*;

**END**;

--PACKAGE BODY

**CREATE OR REPLACE**

**PACKAGE BODY** CHART\_PKG **AS**

**FUNCTION** FUNC\_CHART\_NAME (f\_petid **IN** int)

**RETURN** *varchar2* **AS**

    --NO "DECLARE IN FUNCTIONS JUST STUFF IT IN AFTER AS"

    --USING ANCHORED DATA TYPES TO AVOID ERRORS

        lv\_petfirst PET.PET\_FIRST\_NAME%**TYPE**;

        lv\_petmid PET.PET\_MIDDLE\_NAME%**TYPE**;

        lv\_petlast OWNER.LAST\_NAME%**TYPE**;

        lv\_full\_name *varchar2*(200);

**BEGIN**

**SELECT** PET\_FIRST\_NAME, PET\_MIDDLE\_NAME, OWNER\_LAST\_NAME

**INTO** lv\_petfirst, lv\_petmid, lv\_petlast

**FROM** ANIMAL\_FACTS

**WHERE** PETID **=** f\_petid;

        lv\_full\_name **:=** lv\_petfirst **||**' ' **||** lv\_petmid **||**' '**||** lv\_petlast;

**RETURN** lv\_full\_name;

**EXCEPTION**

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

        DBMS\_OUTPUT.*PUT\_LINE*('This animal id is invalid, are you sure you entered it in correctly?');

**RETURN** NULL;

**END** FUNC\_CHART\_NAME;

**FUNCTION** FUNC\_CHARTHEAD(f\_petid **IN** ANIMAL\_FACTS.PETID%**TYPE**)

**RETURN** *clob*

**AS**

        lv\_animalname *varchar*(3000) **:=** NULL;

        lv\_gender ANIMAL\_GENDER.GENDER\_NAME%**TYPE** **:=** NULL;

        lv\_species\_name ANIMAL\_SPECIES.SPECIES\_NAME%**TYPE** **:=** NULL;

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

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

        lv\_species\_breed *clob* **:=** NULL;

        lv\_age *varchar*(2000) **:=** NULL;

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

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

        lv\_chartline *varchar2*(100) **:=** CHR(13)**||**'-----------------------------------------------'**||**CHR(13);

**BEGIN**

**SELECT** FUNC\_CHART\_NAME(PETID), FUNC\_SPECIES(SPECIESID), FUNC\_BREED(BREEDID), FUNC\_GENDER(GENDERID), TO\_CHAR(TRUNC(MONTHS\_BETWEEN(SYSDATE,BIRTH\_DATE)**/**12, 1)), TO\_CLOB( TEMPERAMENT\_NOTES), FUNC\_ALL\_SIBLING\_BREEDS(PETID)

**INTO** lv\_animalname, lv\_species\_name, lv\_breed\_name, lv\_gender, lv\_age, lv\_temperament, lv\_chartsibs

**FROM** ANIMAL\_FACTS

**WHERE** PETID **=** f\_petid;

        lv\_animalname **:=** 'Animal Name: ' **||** lv\_animalname **||**' ' **||**CHR(13);

        lv\_species\_breed**:=** lv\_gender**||**' : '**||**lv\_species\_name**||**' : '**||** lv\_breed\_name**||**' ' **||**CHR(13);

        lv\_age **:=** 'Age: '**||**lv\_age**||**' years old.'**||**' ' **||**CHR(13);

        lv\_temperament **:=** 'Animal''s general demeanor as observed is: '**||**' ' **||**CHR(13)**||**lv\_temperament**||**' ' **||**CHR(13);

        lv\_charthead **:=** lv\_animalname**||**lv\_species\_breed**||**lv\_age**||**lv\_temperament**||**lv\_chartline**||**lv\_chartsibs**||**lv\_chartline;

**RETURN** lv\_charthead;

**EXCEPTION**

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

        DBMS\_OUTPUT.*PUT\_LINE*('This chart id/pet id does not exist' **||**chr(10)**||**

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

**END** FUNC\_CHARTHEAD;

**FUNCTION** FUNC\_FULLCHARTNOTES(f\_petid **IN** ANIMAL\_FACTS.PETID%**TYPE**)

**RETURN** *clob*

**AS**

        --begin the notes builders

        lv\_clob\_builder *clob***:=** NULL;

        --lv\_encounter\_notes clob:= NULL;

        --lv\_radiol\_notes clob:= NULL;

        --v\_rxhist\_notes clob:= NULL;

        --lv\_prochist\_notes clob:= NULL;

        --lv\_path\_notes clob := NULL;

        lv\_vet *clob* **:=** NULL;

        --column / row export variables

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

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

        --lv\_rx clob:= NULL; PULLING RX OUT FOR NOW, WILL BUILD SEPERATE SERIES OF FUNCTIONS FOR THIS.

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

        lv\_path *clob* **:=** NULL;

        lv\_crit *varchar2*(40) **:=** NULL;

        --CREATE THE CURSOR

**CURSOR** cur\_noteshist **IS**

**SELECT** PETID, VET, EVENT\_DATE, EVENT, CRITDISEASE, NOTES, EVENT\_TYPE

**FROM** CHART\_NOTES\_V

**WHERE** PETID **=** f\_petid

**ORDER BY** EVENT\_DATE;

**BEGIN**

**for** rec\_noteshist **IN** cur\_noteshist **LOOP**

        lv\_encounter **:=** NULL;

        lv\_radiology **:=** NULL;

        lv\_procedure **:=** NULL;

        lv\_path **:=** NULL;

        lv\_crit **:=** NULL ;

        lv\_vet **:=** rec\_noteshist.VET;

**CASE**

**WHEN** rec\_noteshist.EVENT\_TYPE **=** 'PATHOLOGY' **THEN**

**IF** rec\_noteshist.CRITDISEASE **IN** ('y', '1', 'Y')

**THEN** lv\_crit **:=** 'WARNING: Critical Disease Detected';

**ELSE** lv\_crit **:=** NULL;

**END IF**;

                lv\_path **:=** 'On date: '**||**rec\_noteshist.EVENT\_DATE**||** ' by: '**||**lv\_vet**||**' '**||**

                CHR(13)**||** lv\_crit **||**' '**||**

                CHR(13)**||** 'Lab performed: '**||** rec\_noteshist.EVENT**||**

                CHR(13)**||** rec\_noteshist.NOTES **||**CHR(13);

                lv\_clob\_builder**:=** lv\_clob\_builder**||**CHR(13)**||**lv\_path;

**WHEN** rec\_noteshist.EVENT\_TYPE **=** 'CLINICAL\_PROCEDURE' **THEN**

                lv\_procedure **:=** 'On date: '**||**rec\_noteshist.EVENT\_DATE**||** ' by: '**||**lv\_vet**||**' '**||**

                CHR(13)**||** 'Clinical Procedure performed: '**||** rec\_noteshist.EVENT**||**

                CHR(13)**||** rec\_noteshist.NOTES **||**CHR(13);

                lv\_clob\_builder**:=** lv\_clob\_builder**||**CHR(13)**||**lv\_procedure**||**CHR(13);

**WHEN** rec\_noteshist.EVENT\_TYPE **=** 'RADIOLOGY' **THEN**

                lv\_radiology **:=** 'On date: '**||**rec\_noteshist.EVENT\_DATE**||**' A radiological image was taken'**||**

                CHR(13)**||** rec\_noteshist.NOTES **||**

                CHR(13)**||**'Refer to radiology sub-chart for images'**||**CHR(13);

                    lv\_clob\_builder**:=** lv\_clob\_builder**||**CHR(13)**||**lv\_radiology**||**CHR(13);

**WHEN** rec\_noteshist.EVENT\_TYPE **=** 'ENCOUNTER' **THEN**

                lv\_encounter **:=** 'On date: '**||**rec\_noteshist.EVENT\_DATE**||** ' by: '**||**lv\_vet**||**' '**||**

                CHR(13)**||**'Weight Recorded: '**||** rec\_noteshist.EVENT**||**'lbs'**||**

                CHR(13)**||** rec\_noteshist.NOTES **||**CHR(13);

                lv\_clob\_builder**:=** lv\_clob\_builder**||**CHR(13)**||**lv\_encounter**||**CHR(13);

**ELSE**

                lv\_clob\_builder **:=** lv\_clob\_builder **||** CHR(13) **||** 'No further events found';

**END CASE**;

**END LOOP**;

**RETURN** lv\_clob\_builder;

**EXCEPTION**

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

        lv\_clob\_builder**:=** 'No data found';

**END** FUNC\_FULLCHARTNOTES;

**FUNCTION** FUNC\_RX\_CHART\_DETAILS(f\_petid **IN** RX\_HISTORY.PETID%**TYPE**)

**RETURN** *clob*

**AS**

    lv\_petname *varchar2*(1000) **:=** NULL;

    lv\_petgender *varchar2*(1000) **:=** NULL;

    lv\_petbreed *varchar2*(1000) **:=** NULL;

    lv\_petspecies *varchar2*(1000) **:=** NULL;

    lv\_vetname *varchar2*(1000) **:=** NULL;

    lv\_drugname *varchar2*(1000) **:=** NULL;

    lv\_drugdose *varchar2*(1000) **:=** NULL;

    lv\_maint *char*(1) **:=** NULL;

    lv\_unitsrxd int **:=** NULL;

    lv\_tpd int **:=** NULL;

    lv\_datewritten *date* **:=** NULL;

    lv\_notes *clob* **:=** NULL;

    lv\_clinicalevent *varchar2*(2000) **:=** NULL;

    lv\_clob\_body *clob* **:=** NULL;

    lv\_clob\_head *clob* **:=** NULL;

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

    lv\_medlist *clob* **:=** NULL;

    lv\_non\_maint\_count int **:=** NULL;

    lv\_maint\_count int **:=** NULL;

    lv\_liststart *varchar2*(100) **:=** NULL;

    lv\_full\_clob *clob* **:=** NULL;

    lv\_drugname\_m1 *varchar2*(1000) **:=** NULL;

**CURSOR** cur\_rxinfo **IS**

**SELECT** RXID, PET\_NAME, VET\_NAME, DRUGID, DRUG\_NAME, DRUG\_DOSAGE, IS\_MAINTENANCE\_MED, DRUG\_UNITS\_PRESCRIBED, TIMES\_PER\_DAY, DATE\_WRITTEN, NOTES, CLINICAL\_EVENT

**FROM** RX\_DETAILS\_V

**WHERE** PETID **=** f\_petid

**ORDER BY** DATE\_WRITTEN;

**CURSOR** cur\_rxinfo\_dist **IS**

**SELECT** **DISTINCT** r.DRUGID, r.DRUG\_NAME, r.IS\_MAINTENANCE\_MED

**FROM** RX\_DETAILS\_V r **JOIN**

                                (**SELECT** **DISTINCT** DRUGID

**FROM** RX\_DETAILS\_V

**WHERE** PETID **=** f\_petid) v

**ON** r.DRUGID **=** v.DRUGID

**WHERE** PETID **=** f\_petid;

**BEGIN**

    /\*START BUILDING THE HEADER OF THE REPORT \*/

**SELECT** FUNC\_CHART\_NAME(PETID), FUNC\_GENDER(GENDERID), FUNC\_SPECIES(SPECIESID), FUNC\_BREED(BREEDID)

**INTO** lv\_petname, lv\_petgender, lv\_petspecies, lv\_petbreed

**FROM** ANIMAL\_FACTS

**WHERE** PETID **=** f\_petid;

        lv\_petname **:=** 'RX info for animal: '**||**lv\_petname**||**' '**||**lv\_petgender**||**CHR(13)**||**

                     lv\_petspecies**||**' : '**||**lv\_petbreed**||**CHR(13);

    lv\_non\_maint\_count **:=** 0;

    lv\_maint\_count **:=** 0;

**FOR** rec\_rxinfo\_dist **IN** cur\_rxinfo\_dist **LOOP**

            lv\_maint **:=** rec\_rxinfo\_dist.IS\_MAINTENANCE\_MED;

            lv\_drugname\_m1 **:=** rec\_rxinfo\_dist.DRUG\_NAME;

**CASE**

**WHEN** lv\_maint **IN** ('n', 'N','0') **OR** lv\_maint IS NULL **THEN**

                lv\_non\_maint\_count **:=** lv\_non\_maint\_count **+** 1;

                lv\_medlist **:=** lv\_medlist**||**

                CHR(13)**||**lv\_non\_maint\_count **||**'. '**||**lv\_drugname\_m1;

**ELSE**

                    lv\_maint\_count **:=** lv\_maint\_count **+** 1;

                    lv\_maintmedlist **:=** lv\_maintmedlist**||**

                    CHR(13)**||**lv\_maint\_count**||**'. '**||** lv\_drugname\_m1;

**END CASE**;

**END LOOP**;

    lv\_clob\_head **:=** lv\_petname**||**

                    CHR(13)**||**'Currently taking the following medicines listed as maintenance medicines: '**||**

                    CHR(13)**||** lv\_maintmedlist**||**CHR(10)**||**

                    CHR(13)**||** 'List of other known meds prescribed in the past is: '**||**

                    CHR(13)**||** lv\_medlist **||**

                    CHR(13)**||**'----------------------------------------------------------------------------'**||**

                    CHR(13);

    /\*END HEADER BEGIN FULL DETAILS OF MEDS \*/

    lv\_liststart **:=** 'Beginning full historical medicine list';

**FOR** rec\_rxinfo **IN** cur\_rxinfo **LOOP**

            lv\_drugname **:=** rec\_rxinfo.DRUG\_NAME;

            lv\_datewritten **:=** rec\_rxinfo.DATE\_WRITTEN;

            lv\_vetname **:=** rec\_rxinfo.VET\_NAME;

            lv\_drugdose **:=** rec\_rxinfo.DRUG\_DOSAGE;

            lv\_unitsrxd **:=** rec\_rxinfo.DRUG\_UNITS\_PRESCRIBED;

            lv\_tpd **:=** rec\_rxinfo.TIMES\_PER\_DAY;

            lv\_notes **:=** rec\_rxinfo.NOTES;

            lv\_clinicalevent **:=**rec\_rxinfo.CLINICAL\_EVENT;

**IF** lv\_clinicalevent **<>** 'none'

**THEN**

                lv\_clob\_body **:=** lv\_clob\_body**||**

                            CHR(13)**||** 'ON: '**||**lv\_datewritten**||**' VET: '**||**lv\_vetname**||**' prescribed '**||**

                            CHR(13)**||**lv\_drugname**||**' DOSE: '**||**lv\_drugdose**||** 'TPD: '**||** lv\_tpd**||**

                            CHR(13)**||**'As part of clinical procedure: '**||**lv\_clinicalevent**||**

                            CHR(13)**||** 'Notes (if any): '**||**lv\_notes**||**

                            CHR(10);

**ELSE**

                lv\_clob\_body **:=** lv\_clob\_body**||**

                                CHR(13)**||** 'ON: '**||**lv\_datewritten**||**' VET: '**||**lv\_vetname**||**' prescribed '**||**

                                CHR(13)**||**lv\_drugname**||**' DOSE: '**||**lv\_drugdose**||** 'TPD: '**||** lv\_tpd**||**

                                CHR(13)**||** 'Notes (if any): '**||**lv\_notes**||**

                                CHR(10);

**END IF**;

**END LOOP**;

        lv\_full\_clob**:=**lv\_clob\_head**||**CHR(13)**||**lv\_liststart**||**CHR(13)**||**lv\_clob\_body;

**RETURN** lv\_full\_clob;

**END** FUNC\_RX\_CHART\_DETAILS;

**FUNCTION** FUNC\_FLTXTCHART (f\_petid **IN** ANIMAL\_FACTS.PETID%**TYPE**)

**return** *clob*

**AS**

    lv\_clobbuild *clob* **:=** NULL;

    lv\_except *clob* **:=** 'This animal ID does not exist something is wrong, contact your DBA.';

**BEGIN**

    lv\_clobbuild **:=** FUNC\_CHARTHEAD(f\_petid) **||**

                    CHR(13)**||**'============================================================================'**||**

                    CHR(13)**||**'BEGIN RX SECTION'**||**

                    CHR(13)**||**'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**||**

                    CHR(13)**||**FUNC\_RX\_CHART\_DETAILS(f\_petid)**||**

                    CHR(13)**||**'BEGIN CLINICAL NOTES'**||**

                    CHR(13)**||**'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**||**

                    CHR(13)**||**FUNC\_FULLCHARTNOTES(f\_petid)**||**

                    CHR(13)**||**

                    CHR(13)**||**'END CHART';

**RETURN** lv\_clobbuild;

**END**;

**EXCEPTION**

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

**RETURN** lv\_except;

**END** FUNC\_FLTXTCHART;

**END** CHART\_PKG;

##### PAYMENT\_PKG

**CREATE OR REPLACE** **PACKAGE** PAYMENT\_PKG

**AS**

**FUNCTION** ENCRYPT\_CC (pv\_card\_no **IN** *VARCHAR2*) **RETURN** *RAW*;

**FUNCTION** DECRYPT\_CC (pv\_encryptcard\_no **IN** *RAW*, pv\_last4 **IN** *VARCHAR2*) **RETURN** *VARCHAR2*;

**PROCEDURE** INVOICE\_MOVE(pv\_invoiceid **IN** int);

**PROCEDURE** PAYMENT\_SUBMIT(pv\_type\_pay **IN** int, pv\_amt\_paid **IN** *number*, pv\_invoiceid **IN** int, pv\_card\_no **IN** *varchar2* **DEFAULT** NULL, pv\_check\_no **IN** int **DEFAULT** NULL);

**END**;

**CREATE OR REPLACE** **PACKAGE BODY** PAYMENT\_PKG

**AS**

**FUNCTION** ENCRYPT\_CC (pv\_card\_no **IN** *VARCHAR2*)

**RETURN** *RAW*

**IS**

        lv\_key *RAW*(128) **:=** UTL\_RAW.*CAST\_TO\_RAW*('a proper key goes here');

        lv\_ccnum *RAW*(128) **:=** UTL\_RAW.*CAST\_TO\_RAW*(pv\_card\_no);

        lv\_encrypted\_out *RAW*(2048) **:=** NULL;

**BEGIN**

            lv\_encrypted\_out **:=** DBMS\_CRYPTO.*ENCRYPT*(lv\_ccnum, DBMS\_CRYPTO.*DES\_CBC\_PKCS5*, lv\_key);

**RETURN** lv\_encrypted\_out;

**END** ENCRYPT\_CC;

**FUNCTION** DECRYPT\_CC (pv\_encryptcard\_no **IN** *RAW*, pv\_last4 **IN** *VARCHAR2*) **RETURN** *VARCHAR2*

**IS**

        lv\_key *RAW*(128) **:=** UTL\_RAW.*CAST\_TO\_RAW*('a proper key goes here');

        lv\_decrypted\_out *VARCHAR2*(24) **:=** NULL;

        lv\_decrypted\_raw *RAW*(2048);

lv\_combination\_out *VARCHAR2*(24);

**BEGIN**

            lv\_decrypted\_raw **:=** DBMS\_CRYPTO.*DECRYPT*(src **=>** pv\_encryptcard\_no,

                                                    typ **=>** DBMS\_CRYPTO.*DES\_CBC\_PKCS5*,

                                                    key **=>** lv\_key);

        lv\_decrypted\_out **:=** UTL\_RAW.*CAST\_TO\_VARCHAR2*(lv\_decrypted\_raw);

lv\_combination\_out **:=** CONCAT(lv\_decrypted\_out, pv\_last4);

**RETURN** lv\_combination\_out;

**END** DECRYPT\_CC;

**PROCEDURE** INVOICE\_MOVE(pv\_invoiceid **IN** int)

**IS**

**BEGIN**

**INSERT** **INTO** INVOICE\_CLOSED

**SELECT** **\***

**FROM** INVOICE\_OPEN

**WHERE** INVOICEID **=** pv\_invoiceid;

**DELETE** **FROM** INVOICE\_OPEN

**WHERE** INVOICEID **=** pv\_invoiceid;

        COMMIT;

**END** INVOICE\_MOVE;

**PROCEDURE** PAYMENT\_SUBMIT(pv\_type\_pay **IN** int, pv\_amt\_paid **IN** *number*, pv\_invoiceid **IN** int, pv\_card\_no **IN** *varchar2* **DEFAULT** NULL, pv\_check\_no **IN** int **DEFAULT** NULL)

**IS**

        lv\_last4 *VARCHAR2*(4) **:=** NULL;

        lv\_first12 *varchar2*(12) **:=** NULL;

        lv\_encryptcc *RAW*(2048) **:=** NULL;

**BEGIN**

**CASE**

**WHEN** pv\_type\_pay **=** 1 **OR** pv\_type\_pay **=** 4 **THEN**

**INSERT** **INTO** PAYMENT\_VERIFICATION(INVOICEID, PAYMENT\_TYPEID, AMOUNT\_PAID, DATE\_PAID)

**VALUES**(pv\_invoiceid, pv\_type\_pay, pv\_amt\_paid, SYSDATE);

                INVOICE\_MOVE(pv\_invoiceid);

**WHEN** pv\_type\_pay **=** 2 **THEN**

**INSERT** **INTO** PAYMENT\_VERIFICATION(INVOICEID, PAYMENT\_TYPEID, AMOUNT\_PAID, CHECK\_NUMBER, DATE\_PAID)

**VALUES**(pv\_invoiceid, pv\_type\_pay, pv\_amt\_paid, pv\_check\_no, SYSDATE);

                INVOICE\_MOVE(pv\_invoiceid);

**WHEN** pv\_type\_pay **=** 3 **THEN**

                lv\_last4 **:=** SUBSTR(pv\_card\_no, **-**4);

                lv\_first12 **:=** SUBSTR(pv\_card\_no, 0, 12);

                lv\_encryptcc **:=** ENCRYPT\_CC(lv\_first12);

**INSERT** **INTO** PAYMENT\_VERIFICATION(INVOICEID, PAYMENT\_TYPEID, AMOUNT\_PAID, DATE\_PAID, CC\_ENCRYPTED12, CC\_LAST4)

**VALUES**(pv\_invoiceid, pv\_type\_pay, pv\_amt\_paid, SYSDATE, lv\_encryptcc, lv\_last4);

                INVOICE\_MOVE(pv\_invoiceid);

**ELSE** NULL;

**END CASE**;

        COMMIT;

**END** PAYMENT\_SUBMIT;

**END**;

### VARIOUS SQL SNIPPETS

#### LOB SEARCHING

**select** **\***

**from** SPLIT\_TEXT\_CHART\_LVNG\_V

**where** dbms\_lob.*instr*(CHART\_NOTES,TO\_CLOB('FIV')) **>** 0;

#### STAFF TABLE DATA LOADING

--------------------------------------------------------

-- File created - Saturday-June-23-2018

--------------------------------------------------------

REM INSERTING into STAFF

**SET** **DEFINE** OFF;

**INSERT** **INTO** STAFF (STAFF\_FIRST\_NAME,STAFF\_LAST\_NAME,EMPLOYMENT\_DATE,TERMINATION\_DATE,IS\_REHIREABLE,IS\_VET,DATABASE\_ROLE) **VALUES** ('Javier','Ramos',to\_date('12-OCT-04','DD-MON-RR'),null,null,'0',null);

**INSERT** **INTO** STAFF (STAFF\_FIRST\_NAME,STAFF\_LAST\_NAME,EMPLOYMENT\_DATE,TERMINATION\_DATE,IS\_REHIREABLE,IS\_VET,DATABASE\_ROLE) **VALUES** ('Emma','Summers',to\_date('03-MAR-11','DD-MON-RR'),null,null,'1',null);

**INSERT** **INTO** STAFF (STAFF\_FIRST\_NAME,STAFF\_LAST\_NAME,EMPLOYMENT\_DATE,TERMINATION\_DATE,IS\_REHIREABLE,IS\_VET,DATABASE\_ROLE) **VALUES** ('Ororo','Rogers',null,null,null,'Y',null);

**INSERT** **INTO** STAFF (STAFF\_FIRST\_NAME,STAFF\_LAST\_NAME,EMPLOYMENT\_DATE,TERMINATION\_DATE,IS\_REHIREABLE,IS\_VET,DATABASE\_ROLE) **VALUES** ('Eric','Xavier',null,null,null,'y',null);

**INSERT** **INTO** STAFF (STAFF\_FIRST\_NAME,STAFF\_LAST\_NAME,EMPLOYMENT\_DATE,TERMINATION\_DATE,IS\_REHIREABLE,IS\_VET,DATABASE\_ROLE) **VALUES** ('Dave','Babler',to\_date('20-MAY-18', 'DD-MON-RR',null,null,'y',null);

#### INDEX CREATION

**CREATE** INDEX DRUGNAME\_RXHIST\_IDX

**ON** RX\_HISTORY(FUNC\_DRUGNAME(DRUGID));

**CREATE** INDEX AN\_FCTS\_CHRT\_NM\_IDX

**ON** ANIMAL\_FACTS(CHART\_PKG.*FUNC\_CHART\_NAME*(PETID));

#### DIRECTORY CREATION & BFILE LOADS

**CREATE OR REPLACE** DIRECTORY

RADIMG

**AS**

    'D:/Put\_Stuff\_Here/Images';

**UPDATE** RADIOLOGY\_HISTORY

**SET** RADIMG\_FILES **=** BFILENAME('RADIMG', 'xray\_snek.jpg')

**WHERE** RADIMGID **=** 1;

**UPDATE** RADIOLOGY\_HISTORY

**SET** RADIMG\_FILES **=** BFILENAME('RADIMG', 'dog\_Brain-MRI.jpeg')

**WHERE** RADIMGID **=** 2;

**UPDATE** RADIOLOGY\_HISTORY

**SET** RADIMG\_FILES **=** BFILENAME('RADIMG', 'feline\_PET SCAN.jpg')

**WHERE** RADIMGID **=** 3;

**UPDATE** RADIOLOGY\_HISTORY

**SET** RADIMG\_FILES **=** BFILENAME('RADIMG', 'monitorlizard\_catscan.jpg')

**WHERE** RADIMGID **=** 4;

**CREATE OR REPLACE** DIRECTORY

IMPORTCHART

**AS**

    'D:/Put\_Stuff\_Here/PDF/';

#### JOBS

**BEGIN**

DBMS\_SCHEDULER.*CREATE\_JOB*(

    job\_name **=>** 'Expire\_Estimates',

    job\_type **=>** 'STORED\_PROCEDURE',

    job\_class **=>** 'FEES\_ESTIMATEEXPIRE',

    job\_action **=>** 'PROC\_EXPIRE\_EST',

    repeat\_interval **=>** 'freq=daily; byhour=6; byminute=0; bysecond=0;',

    enabled **=>** TRUE);

**END**;

**BEGIN**

DBMS\_SCHEDULER.*CREATE\_JOB*(

    job\_name **=>** 'Add\_Late\_Fees',

    job\_type **=>** 'STORED\_PROCEDURE',

    job\_class **=>** 'FEES\_ESTIMATEEXPIRE',

    job\_action **=>** 'PROC\_LATE\_FEE',

    repeat\_interval **=>** 'freq=daily; byhour=6; byminute=0; bysecond=0;',

    enabled **=>** TRUE);

**END**;

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: Right Click + Open Link from the popup menu will pop this document’s images out in a browser window [↑](#footnote-ref-8)