VETERINARY DATABASE: REQUIREMENTS IMPLEMENTATION & PROOF OF CONCEPT

Dave Babler

Supervised by: Professor Larry Bross

TABLE OF CONTENTS

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.

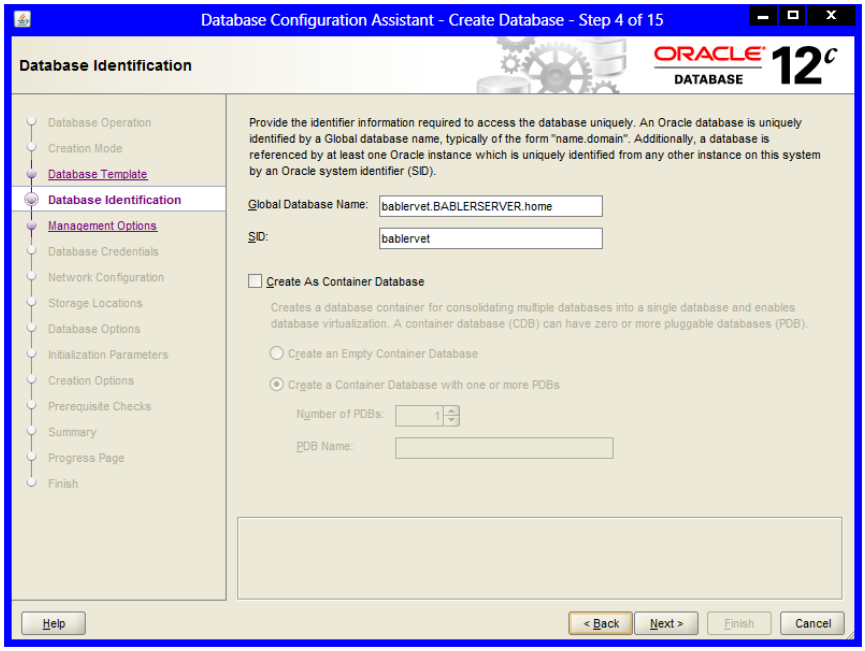


Figure 1

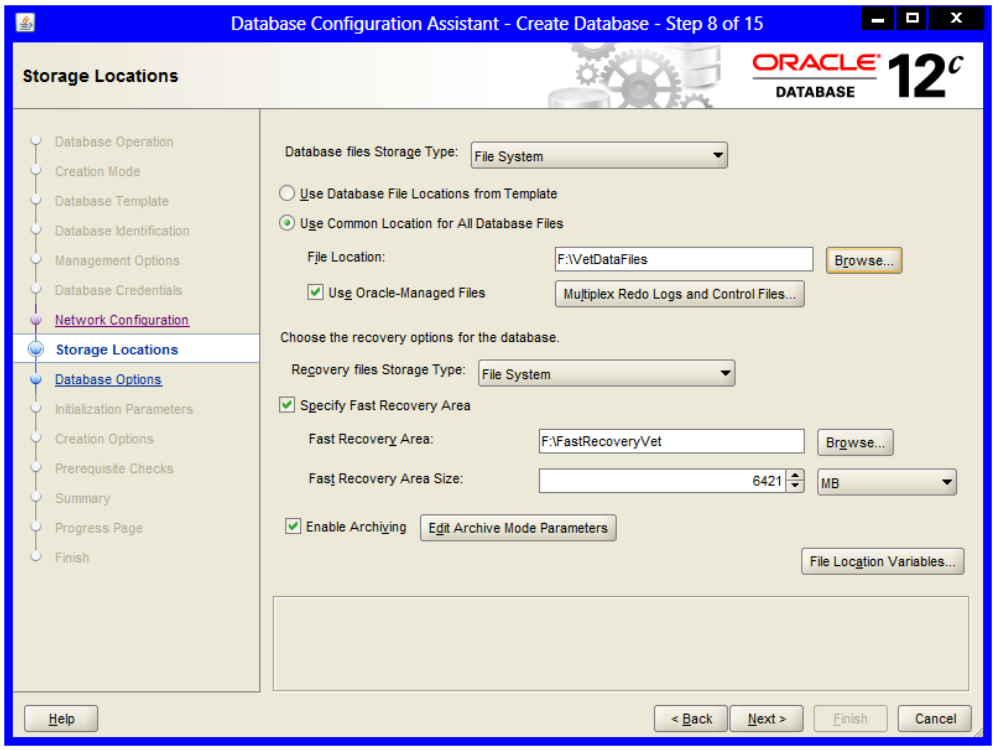


Figure 2

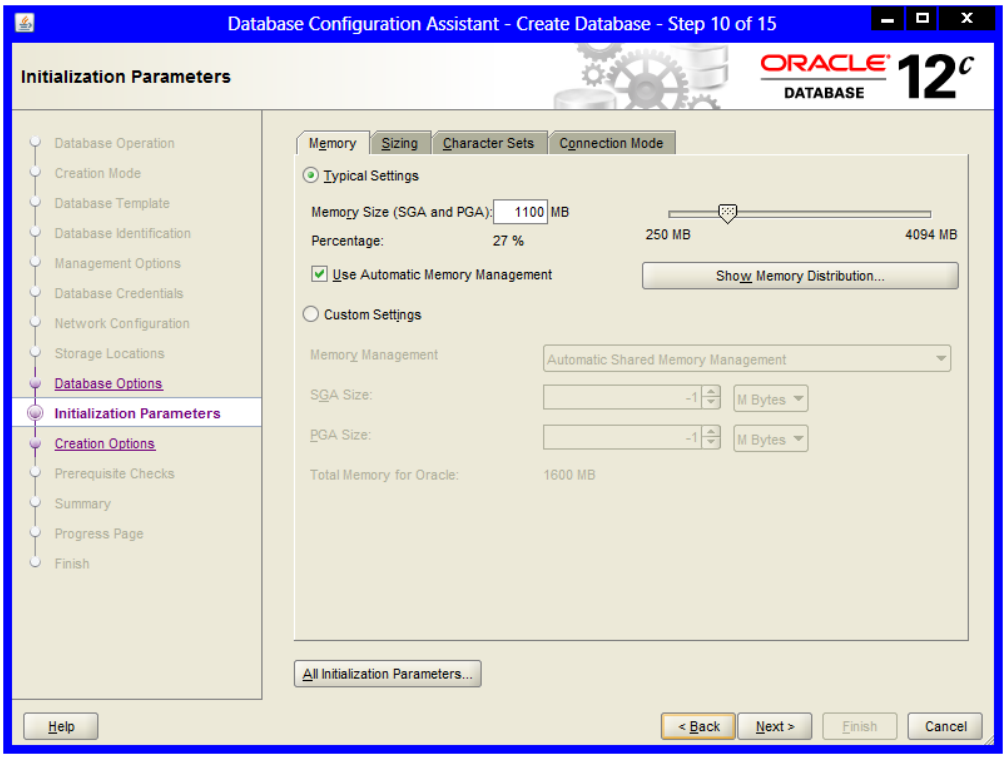


Figure 3

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].

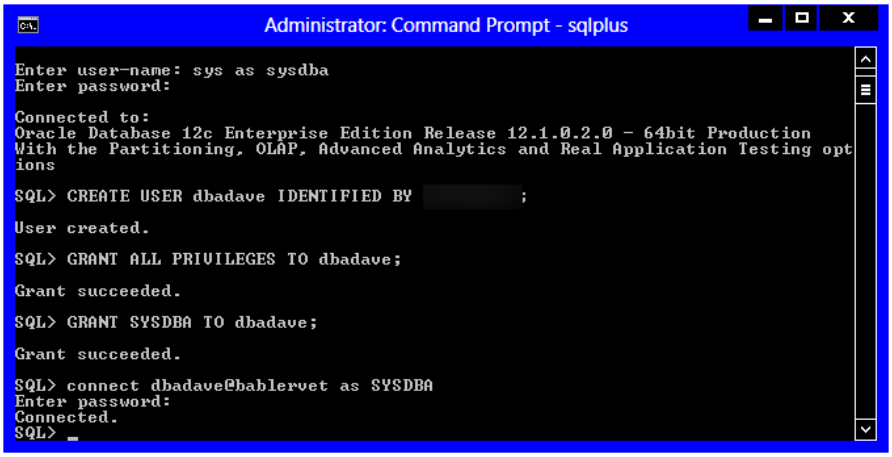


Figure 4

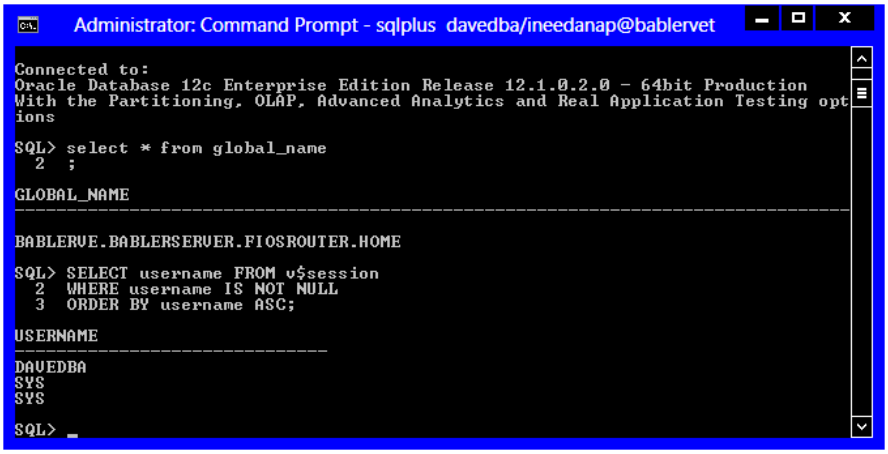


Figure 5

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.

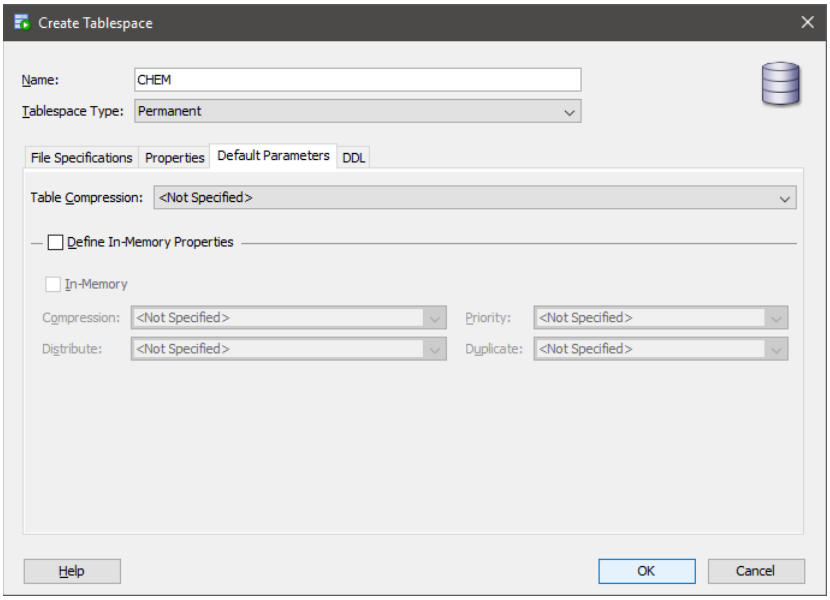


Figure 6

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.

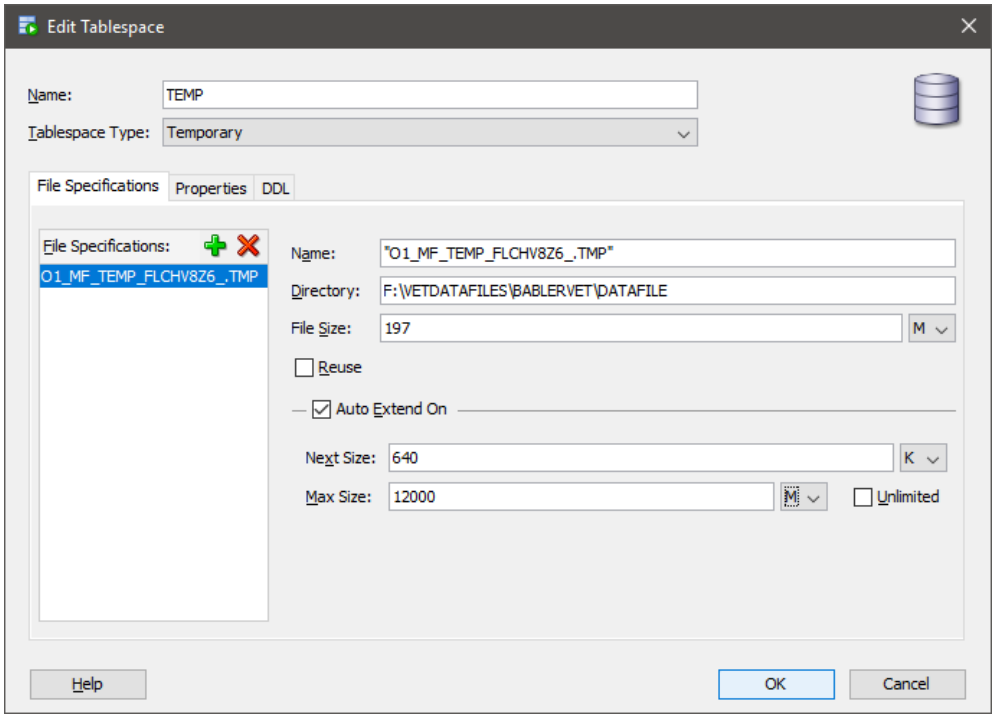


Figure 8

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].

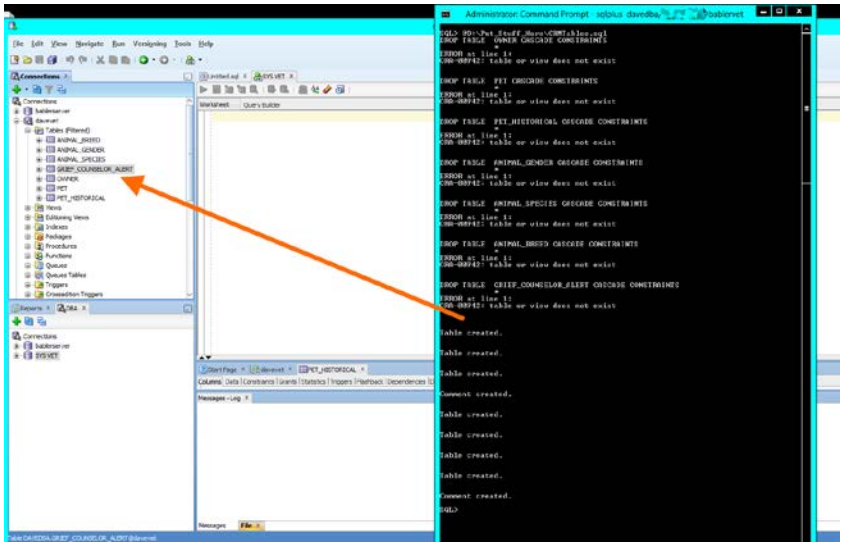


Figure 9

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

```
--RADIOLOGY_HISTORY
CREATE TABLE RADIOLOGY_HISTORY(
  PetID int,
  RadImgID  int,
  RadImg_Date_Taken date,
  RadImg_Notes  clob,
  RadImg_Files  bfile,
  --Composite primary key syntax below
  CONSTRAINT pk_petid_rad PRIMARY KEY (PetID, RadimgID),
  --End composite primary key syntax
  CONSTRAINT fk_petid_rad FOREIGN KEY (PetID)
      REFERENCES ANIMAL_FACTS(PetID))

TABLESPACE CHART;
```

```
Script Output
Task completed in 0.088 seconds
Table RADIOLOGY_HISTORY created.
```

Figure 10

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

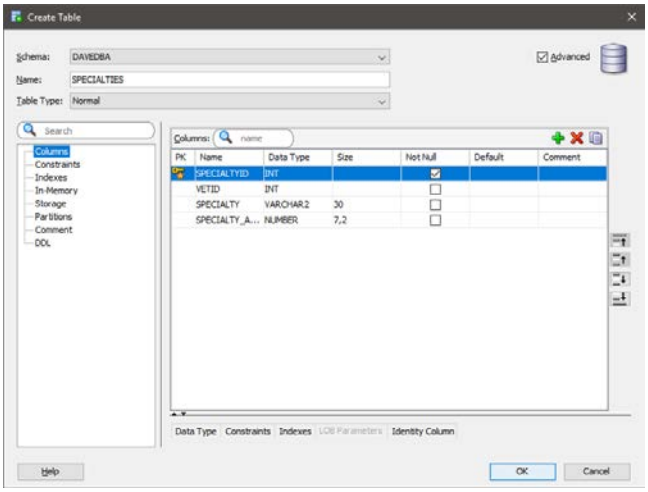


Figure 11

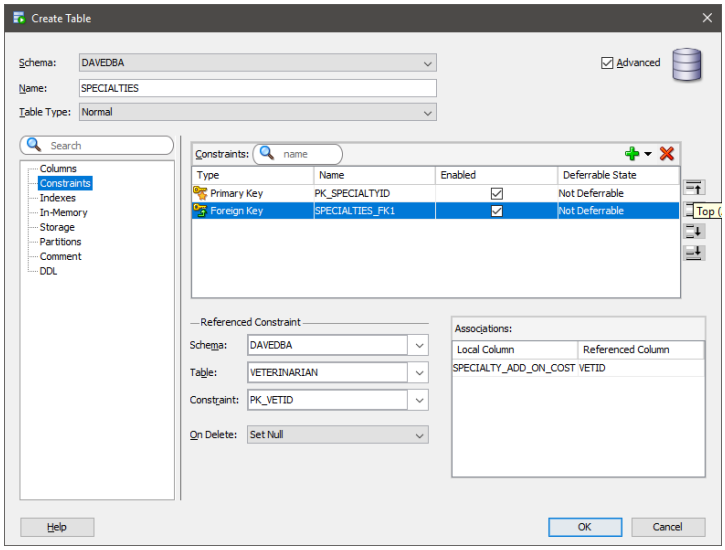


Figure 12

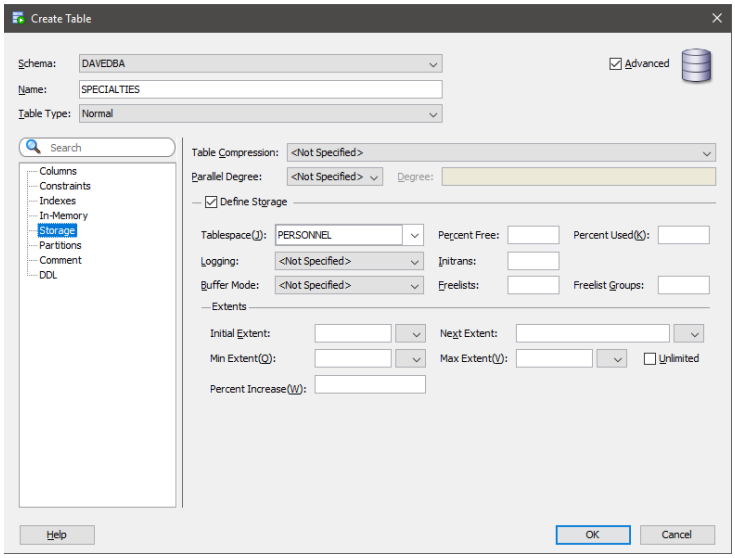


Figure 13

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].

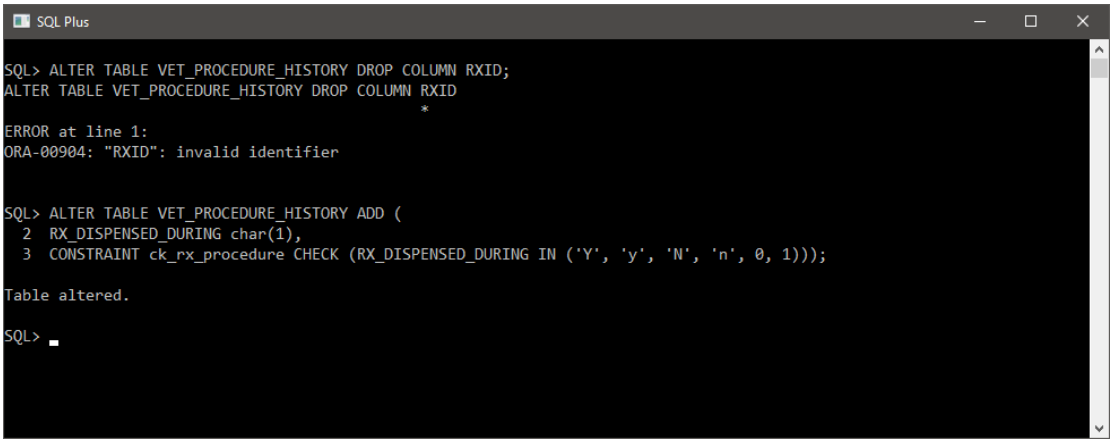


Figure 14

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].

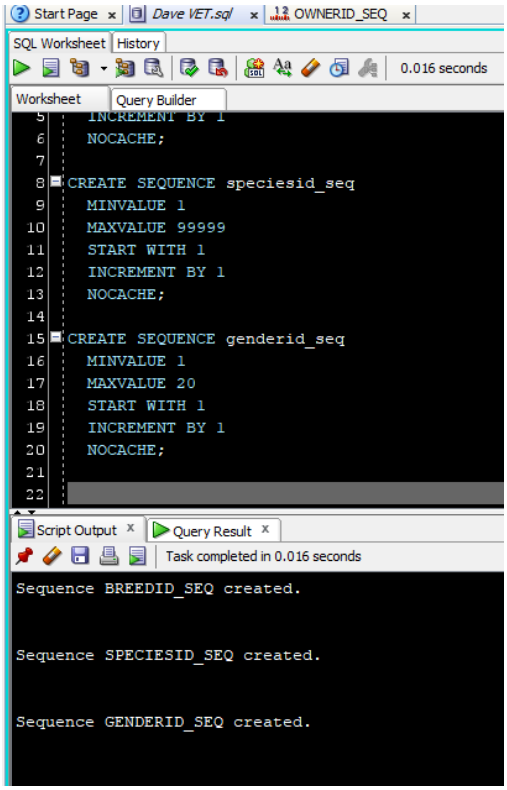


Figure 16

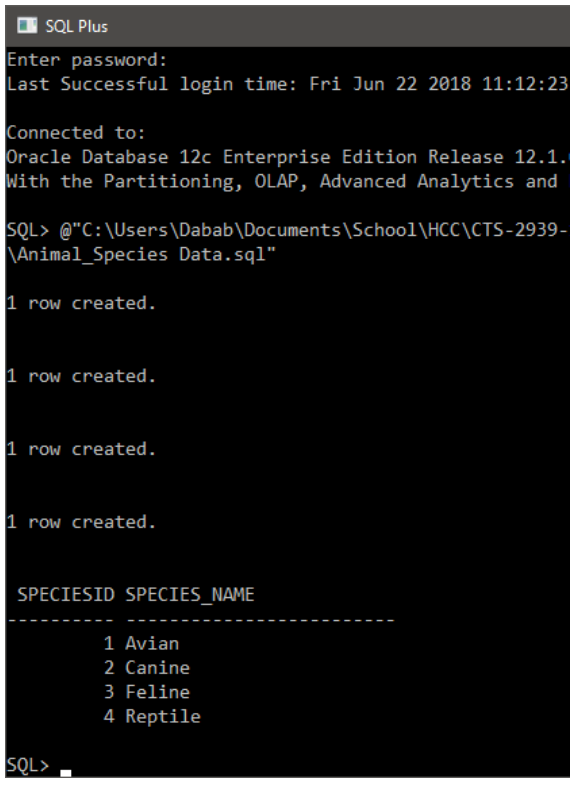


Figure 16

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].

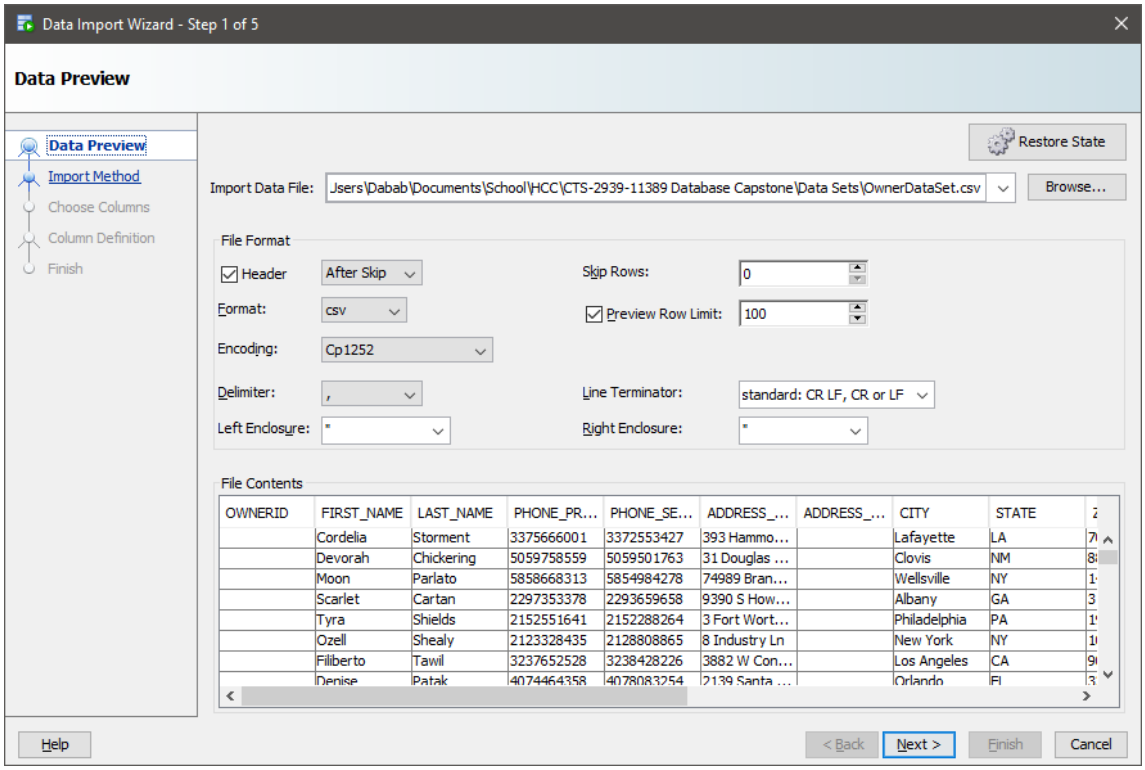


Figure 17

BUSINESS RULES IMPLEMENTATION[1]

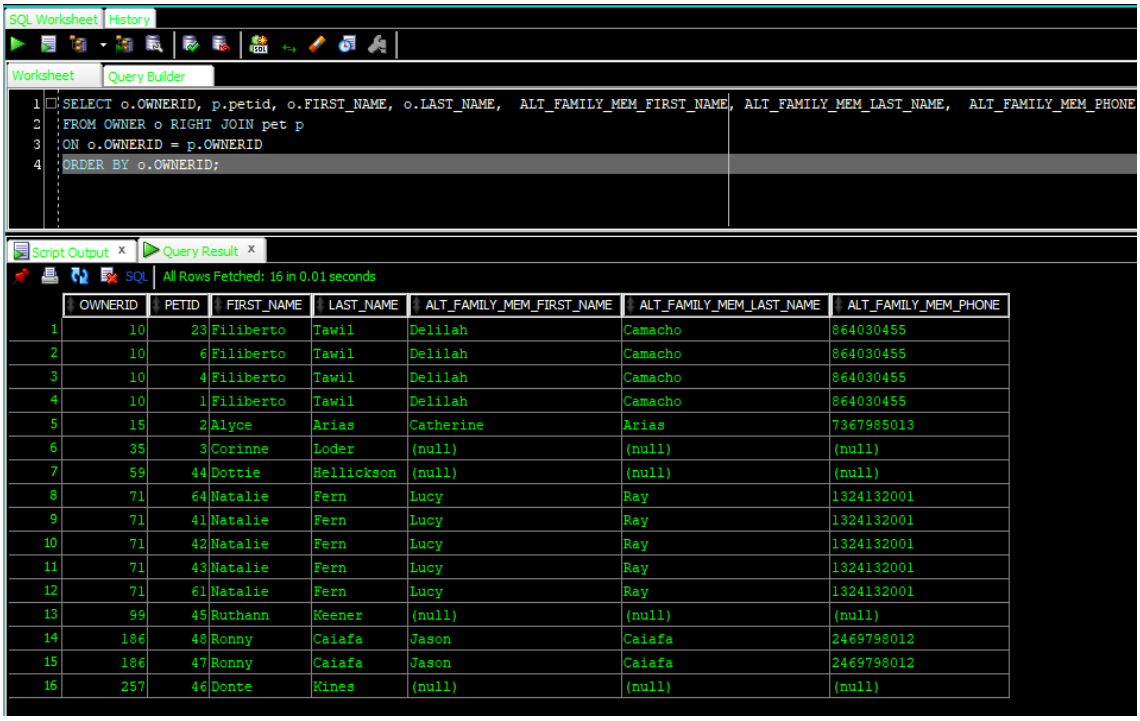
RECORD KEEPING, RECEPTION, CUSTOMER RELATIONS MANAGMENT

CRM-01 AN OWNER MAY HAVE MANY PETS; A PET MAY ONLY BELONG TO ONE OWNER.
CRM-02[2] 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

[1] Full text definitions and a table of all business rules, transactions and report requirements can be found in CSFDAVD.
[2] Business rules will be grouped however, they may not necessarily be presented in numerical order.



	OWNERID	PETID	FIRST_NAME	LAST_NAME	ALT_FAMILY_MEM_FIRST_NAME	ALT_FAMILY_MEM_LAST_NAME	ALT_FAMILY_MEM_PHONE
1	10	23	Filiberto	Tawil	Delilah	Camacho	864030455
2	10	6	Filiberto	Tawil	Delilah	Camacho	864030455
3	10	4	Filiberto	Tawil	Delilah	Camacho	864030455
4	10	1	Filiberto	Tawil	Delilah	Camacho	864030455
5	15	2	Alyce	Arias	Catherine	Arias	7367985013
6	35	3	Corinne	Loder	(null)	(null)	(null)
7	59	44	Dottie	Hellickson	(null)	(null)	(null)
8	71	64	Natalie	Fern	Lucy	Ray	1324132001
9	71	41	Natalie	Fern	Lucy	Ray	1324132001
10	71	42	Natalie	Fern	Lucy	Ray	1324132001
11	71	43	Natalie	Fern	Lucy	Ray	1324132001
12	71	61	Natalie	Fern	Lucy	Ray	1324132001
13	99	45	Ruthann	Keener	(null)	(null)	(null)
14	186	48	Ronny	Caiafa	Jason	Caiafa	2469798012
15	186	47	Ronny	Caiafa	Jason	Caiafa	2469798012
16	257	46	Donte	Kines	(null)	(null)	(null)

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].

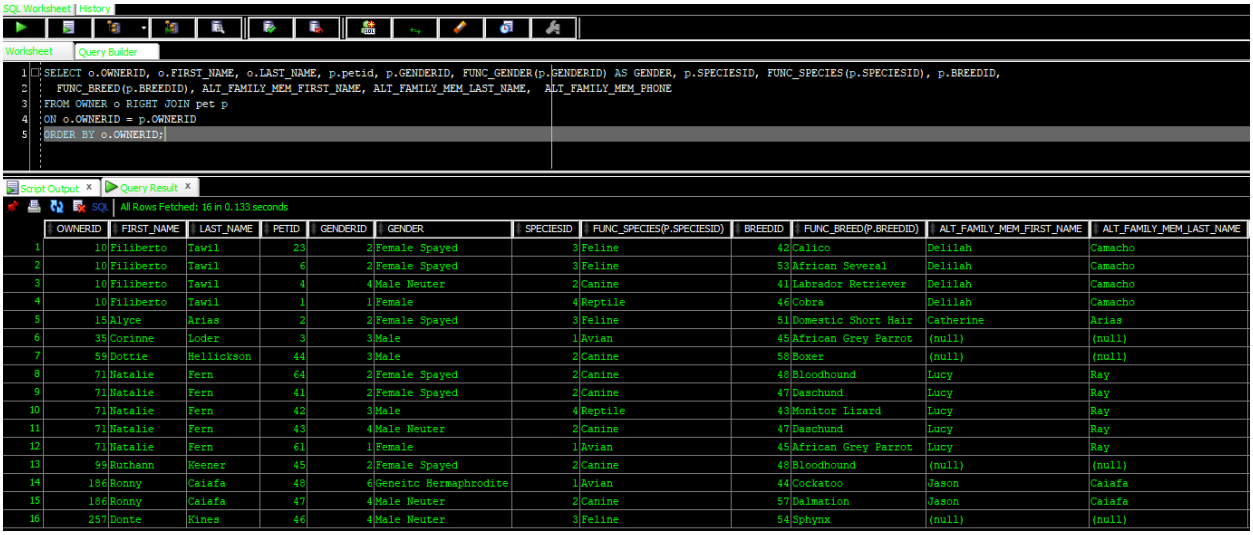
```
CREATE OR REPLACE FUNCTION FUNC_BREED(
       f_breedid IN ANIMAL_BREED.BREEDID%TYPE)
RETURN varchar2
AS
lv_breed_name ANIMAL_BREED.BREED_NAME%TYPE;
lv_except varchar2(100) 'No Breed Found';


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



```
SELECT o.OWNERID, o.FIRST_NAME, o.LAST_NAME, p.petid, p.GENDERID, FUNC_GENDER(p.GENDERID) AS GENDER, p.SPECIESID, FUNC_SPECIES(p.SPECIESID), p.BREEDID,
  FUNC_BREED(p.BREEDID), ALT_FAMILY_MEM_FIRST_NAME, ALT_FAMILY_MEM_LAST_NAME,  ALT_FAMILY_MEM_PHONE
FROM OWNER o RIGHT JOIN pet p
ON o.OWNERID = p.OWNERID
ORDER BY o.OWNERID;
```

All Rows Fetched: 16 in 0.133 seconds

	OWNERID	FIRST_NAME	LAST_NAME	PETID	GENDERID	GENDER	SPECIESID	FUNC_SPECIES(P.SPECIESID)	BREEDID	FUNC_BREED(P.BREEDID)	ALT_FAMILY_MEM_FIRST_NAME	ALT_FAMILY_MEM_LAST_NAME
1	10	Filiberto	Tawil	23	2	Female Spayed	3	Feline	42	Calico	Delilah	Camacho
2	10	Filiberto	Tawil	6	2	Female Spayed	3	Feline	53	African Several	Delilah	Camacho
3	10	Filiberto	Tawil	4	4	Male Neuter	2	Canine	41	Labrador Retriever	Delilah	Camacho
4	10	Filiberto	Tawil	1	1	Female	4	Reptile	46	Cobra	Delilah	Camacho
5	15	Alyce	Arias	2	2	Female Spayed	3	Feline	51	Domestic Short Hair	Catherine	Arias
6	35	Corinne	Loder	3	3	Male	1	Avian	45	African Grey Parrot	(null)	(null)
7	59	Dottie	Hellickson	44	3	Male	2	Canine	58	Boxer	(null)	(null)
8	71	Natalie	Fern	64	2	Female Spayed	2	Canine	48	Bloodhound	Lucy	Ray
9	71	Natalie	Fern	41	2	Female Spayed	2	Canine	47	Daschund	Lucy	Ray
10	71	Natalie	Fern	42	3	Male	4	Reptile	43	Monitor Lizard	Lucy	Ray
11	71	Natalie	Fern	43	4	Male Neuter	2	Canine	47	Daschund	Lucy	Ray
12	71	Natalie	Fern	61	1	Female	1	Avian	45	African Grey Parrot	Lucy	Ray
13	99	Ruthann	Keener	45	2	Female Spayed	2	Canine	48	Bloodhound	(null)	(null)
14	186	Ronny	Caiafa	48	6	Geneitc Hermaphrodite	1	Avian	44	Cockatoo	Jason	Caiafa
15	186	Ronny	Caiafa	47	4	Male Neuter	2	Canine	57	Dalmation	Jason	Caiafa
16	257	Donte	Kines	46	4	Male Neuter	3	Feline	54	Sphynx	(null)	(null)

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].

	OWNERID	PET_FIRST_NAME	PET_MIDDLE_NAME	LAST_NAME	SPECIES_NAME	BREED_NAME	Patient's Animal Siblings
1	10	Diane	Michelle	Tawil	Feline	African Several	Sara, Steve, Jubilee
2	10	Sara	Snek	Tawil	Reptile	Cobra	Steve, Diane, Jubilee
3	10	Jubilee	(null)	Tawil	Feline	Calico	Sara, Steve, Diane
4	10	Steve	Cap	Tawil	Canine	Labrador Retriever	Sara, Diane, Jubilee
5	15	Jessie	(null)	Arias	Feline	Domestic Short Hair	(null)
6	35	Larry	Squakers	Loder	Avian	African Grey Parrot	(null)
7	59	Jason	(null)	Hellickson	Canine	Boxer	(null)
8	71	Jeffery	(null)	Fern	Canine	Daschund	Suzie, Marisol, Jefferson
9	71	Suzie	(null)	Fern	Canine	Daschund	Marisol, Jeffery, Jefferson
10	71	Jefferson	Twilight	Fern	Avian	African Grey Parrot	Suzie, Marisol, Jeffery
11	71	Marisol	(null)	Fern	Reptile	Monitor Lizard	Suzie, Jeffery, Jefferson
12	99	Hillary	(null)	Keener	Canine	Bloodhound	(null)
13	186	Rover	(null)	Caiafa	Canine	Dalmation	Tweety
14	186	Tweety	(null)	Caiafa	Avian	Cockatoo	Rover

Figure 20

The combination of this code along with the code that satisfied CRM-07 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.
```

[3] I strongly recommend right clicking and then choosing open link to see images in their full size in a browser.

```
        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].

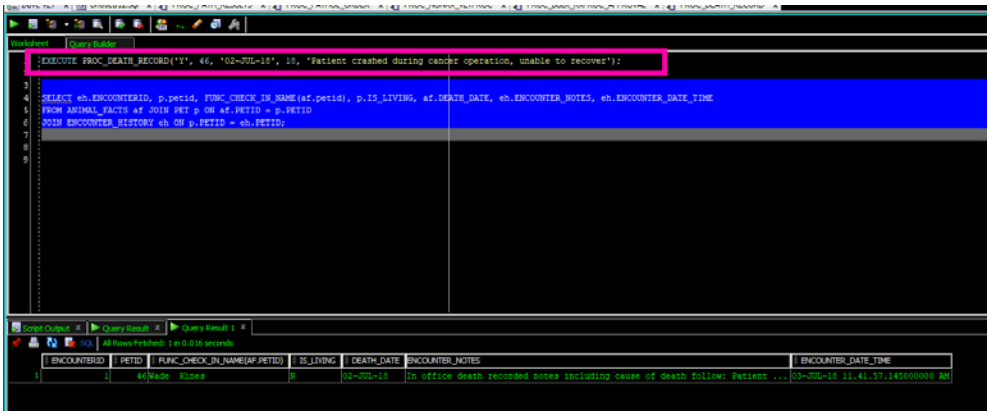


Figure 21

```
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].

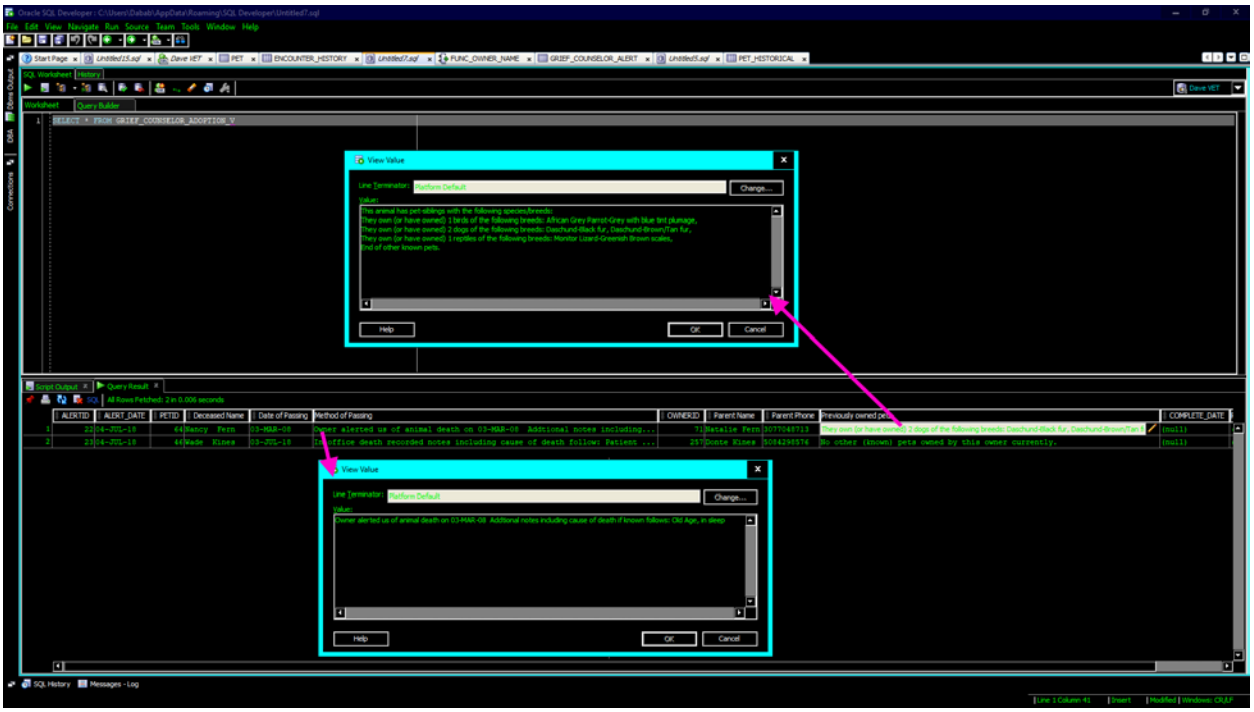
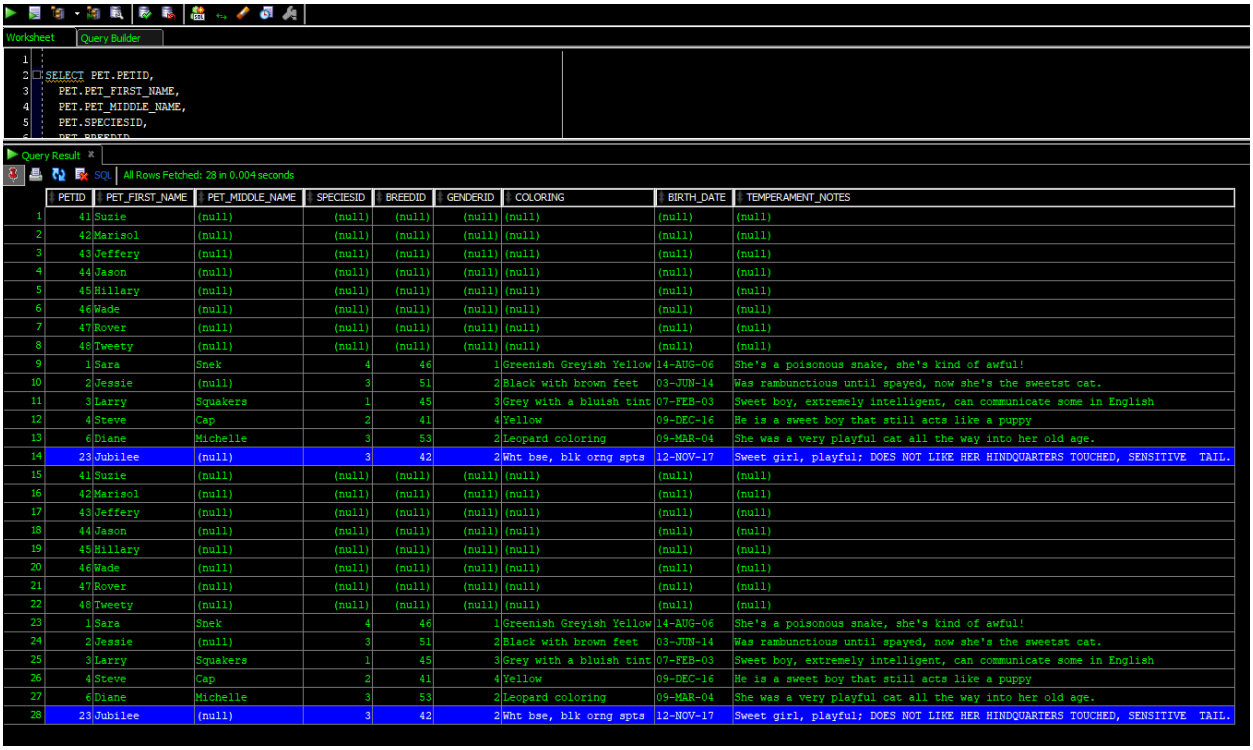


Figure 22

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.



```
SELECT PET.PETID,
    PET.PET_FIRST_NAME,
    PET.PET_MIDDLE_NAME,
    PET.SPECIESID,
```

Query Result — All Rows Fetched: 28 in 0.004 seconds

	PETID	PET_FIRST_NAME	PET_MIDDLE_NAME	SPECIESID	BREEDID	GENDERID	COLORING	BIRTH_DATE	TEMPERAMENT_NOTES
1	41	Suzie	(null)	(null)	(null)	(null)	(null)	(null)	(null)
2	42	Marisol	(null)	(null)	(null)	(null)	(null)	(null)	(null)
3	43	Jeffery	(null)	(null)	(null)	(null)	(null)	(null)	(null)
4	44	Jason	(null)	(null)	(null)	(null)	(null)	(null)	(null)
5	45	Hillary	(null)	(null)	(null)	(null)	(null)	(null)	(null)
6	46	Wade	(null)	(null)	(null)	(null)	(null)	(null)	(null)
7	47	Rover	(null)	(null)	(null)	(null)	(null)	(null)	(null)
8	48	Tweety	(null)	(null)	(null)	(null)	(null)	(null)	(null)
9	1	Sara	Snek	4	46	1	Greenish Greyish Yellow	14-AUG-06	She's a poisonous snake, she's kind of awful!
10	2	Jessie	(null)	3	51	2	Black with brown feet	03-JUN-14	Was rambunctious until spayed, now she's the sweetst cat.
11	3	Larry	Squakers	1	45	3	Grey with a bluish tint	07-FEB-03	Sweet boy, extremely intelligent, can communicate some in English
12	4	Steve	Cap	2	41	4	Yellow	09-DEC-16	He is a sweet boy that still acts like a puppy
13	6	Diane	Michelle	3	53	2	Leopard coloring	09-MAR-04	She was a very playful cat all the way into her old age.
14	23	Jubilee	(null)	3	42	2	Wht bse, blk orng spts	12-NOV-17	Sweet girl, playful; DOES NOT LIKE HER HINDQUARTERS TOUCHED, SENSITIVE TAIL.
15	41	Suzie	(null)	(null)	(null)	(null)	(null)	(null)	(null)
16	42	Marisol	(null)	(null)	(null)	(null)	(null)	(null)	(null)
17	43	Jeffery	(null)	(null)	(null)	(null)	(null)	(null)	(null)
18	44	Jason	(null)	(null)	(null)	(null)	(null)	(null)	(null)
19	45	Hillary	(null)	(null)	(null)	(null)	(null)	(null)	(null)
20	46	Wade	(null)	(null)	(null)	(null)	(null)	(null)	(null)
21	47	Rover	(null)	(null)	(null)	(null)	(null)	(null)	(null)
22	48	Tweety	(null)	(null)	(null)	(null)	(null)	(null)	(null)
23	1	Sara	Snek	4	46	1	Greenish Greyish Yellow	14-AUG-06	She's a poisonous snake, she's kind of awful!
24	2	Jessie	(null)	3	51	2	Black with brown feet	03-JUN-14	Was rambunctious until spayed, now she's the sweetst cat.
25	3	Larry	Squakers	1	45	3	Grey with a bluish tint	07-FEB-03	Sweet boy, extremely intelligent, can communicate some in English
26	4	Steve	Cap	2	41	4	Yellow	09-DEC-16	He is a sweet boy that still acts like a puppy
27	6	Diane	Michelle	3	53	2	Leopard coloring	09-MAR-04	She was a very playful cat all the way into her old age.
28	23	Jubilee	(null)	3	42	2	Wht bse, blk orng spts	12-NOV-17	Sweet girl, playful; DOES NOT LIKE HER HINDQUARTERS TOUCHED, SENSITIVE TAIL.

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].

```
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;
```

[4] Unless noteworthy no further code will be directly embedded into this document; however, all code will be attached as an appendix.

```
                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] 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

[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.

VET_PROCEDURE_HISTORY[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.

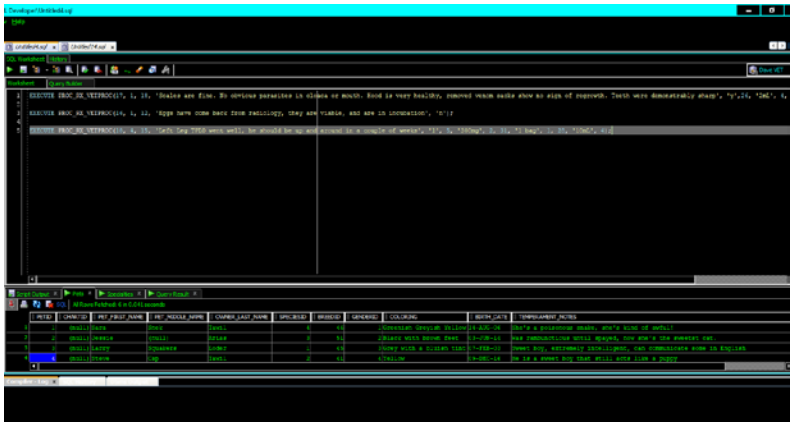


Figure 24

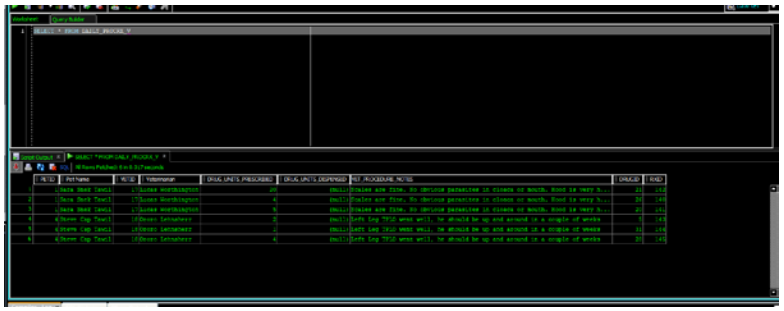


Figure 25

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].

[6] Reminder: all images are referenced to an online version and can be viewed in a browser by clicking <Ctrl> + Click in Microsoft Word.

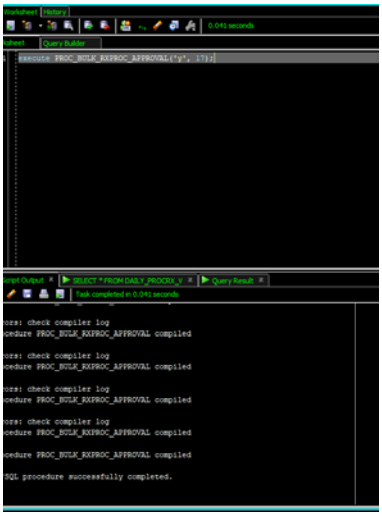


Figure 27

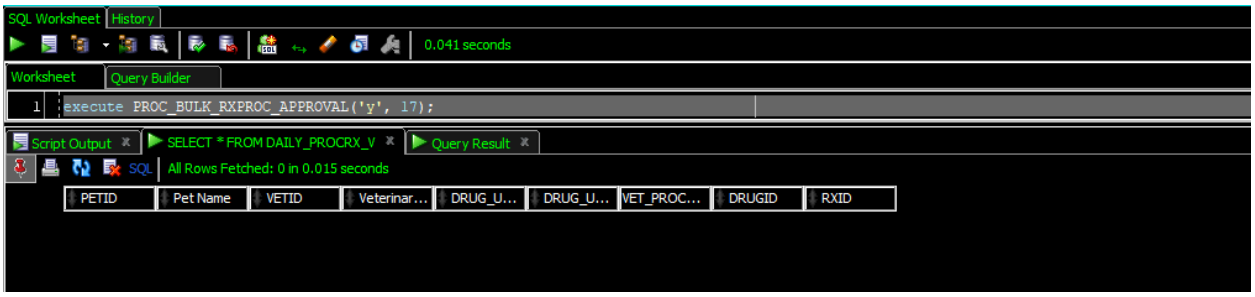


Figure 28

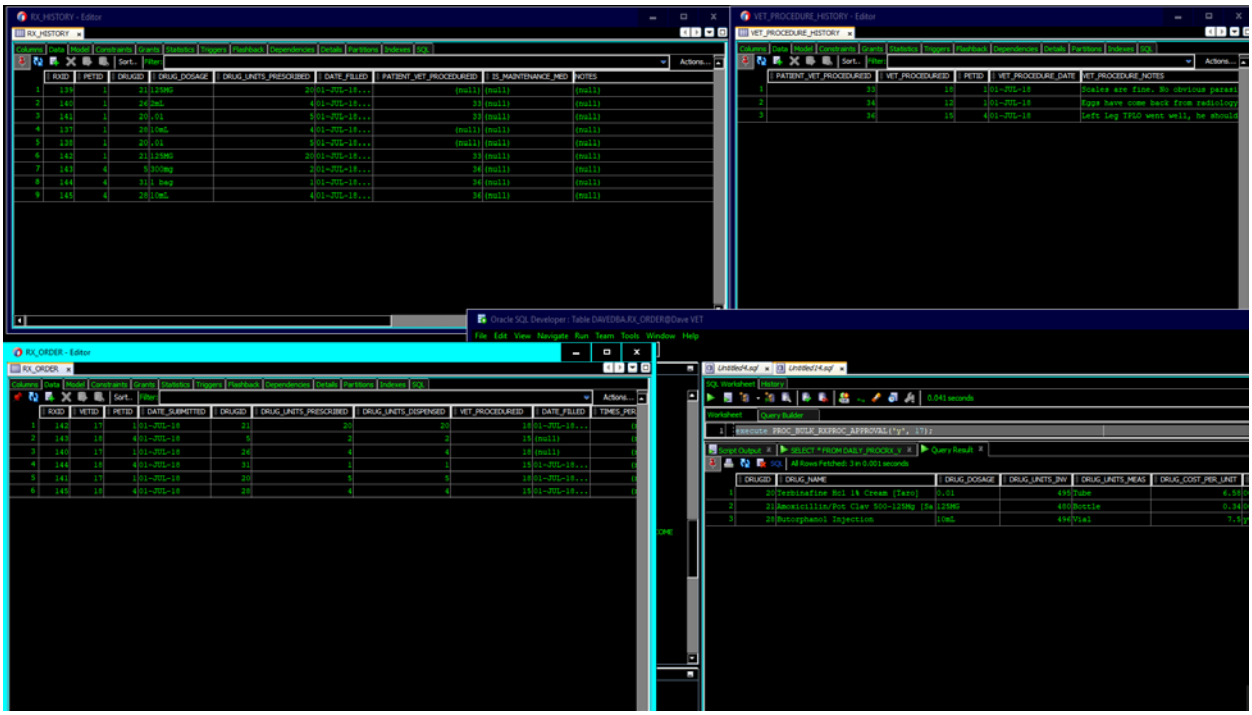


Figure 29

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].

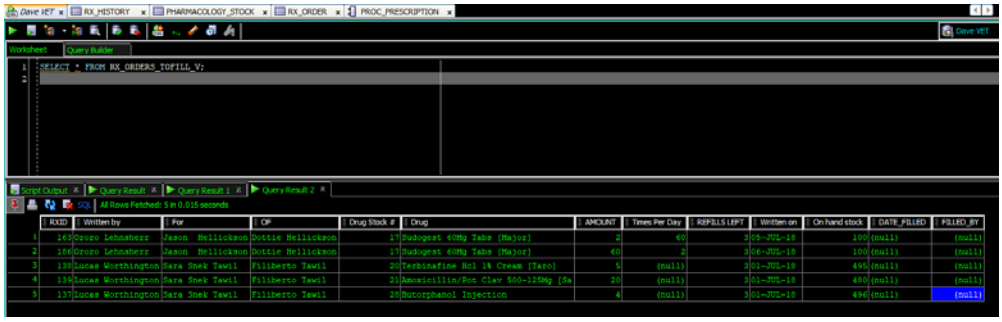


Figure 30

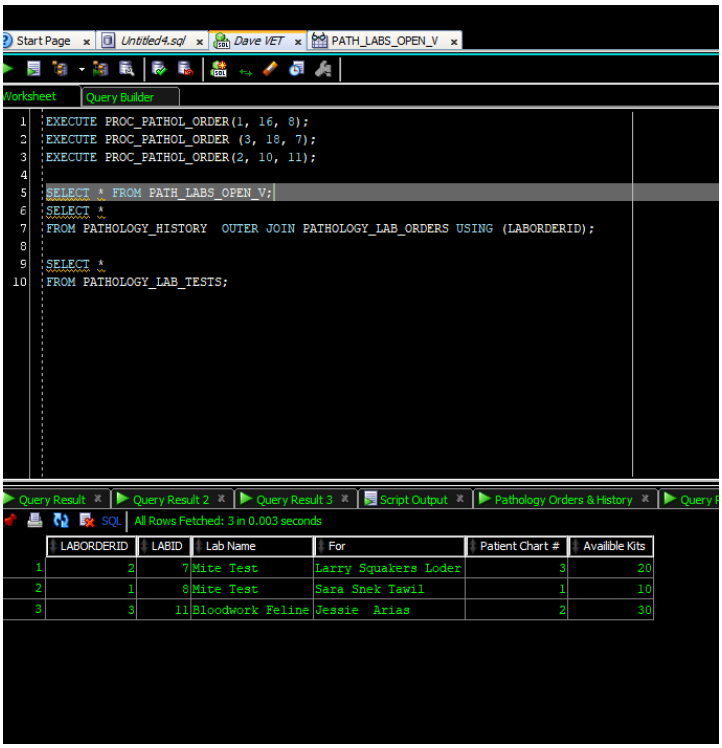


Figure 31

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

- A procedure for a vet to place a prescription [PROC_PRESCRIPTION].
- A procedure for a vet to place a pathology lab order [PROC_PATHOL_ORDER] [Figure 31].
- Functions that show verbose lab names, vet names, and chart names (humans don't respond well to "Fill prescription for patient 10, from vet 15, using drug 98").
- A procedure to complete a pathology lab order [PROC_PATH_RESULTS] [Figure 32].
  - Folded into this program unit is a way of informing veterinarians if pathology has detected a critical illness (such as FIV, malignant tumors, avian-flu, etc.).
  - Also folded into this logic is a way of subtracting lab-kits from inventor [Figure 33].
- A procedure to fill a prescription [PROC_RX_FILL] that:
  - Does the appropriate insertions/updates on the appropriate tables (RX_HISTORY, RX_REFILLS, etc.).
  - Subtracts the drug units from inventory and prevents a ℞ from being filled if there is not enough in stock. Accomplished by using a sub-unit procedure [PROC_DRG_STOCK].
  - Sends out exceptions, notes the account, and prevents transaction completion, if the RX_ORDER table constraint CHK_DISPENSE is violated (you cannot fill a ℞ for more medicine than what is prescribed).
  - Sends out exceptions, notes the account, and prevents transaction completion, if the RX_ORDER table constraint CHK_CONTR is violated (no controlled substance ℞ can be filled for more than 14 days at a time).
  - Sends out exceptions, notes the account, and prevents transaction completion if a vet accidentally submits the same prescription (drug, drug dose, times per day) twice in one day [Figure 34]

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

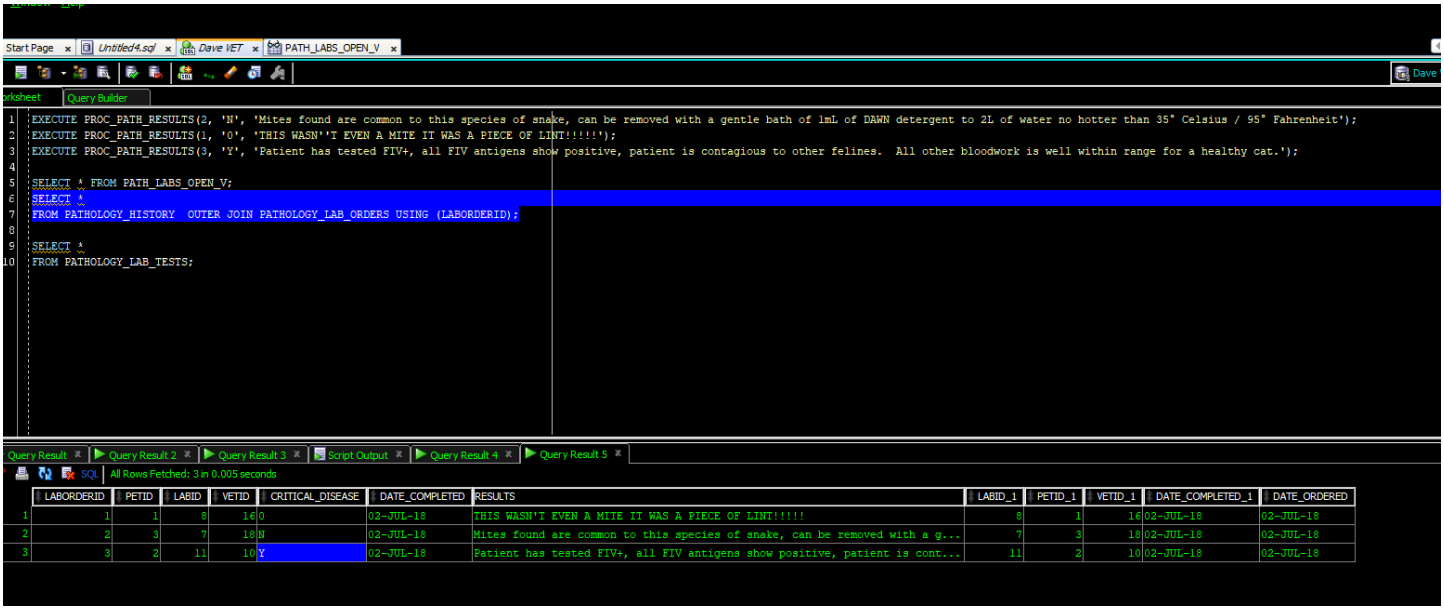


Figure 32

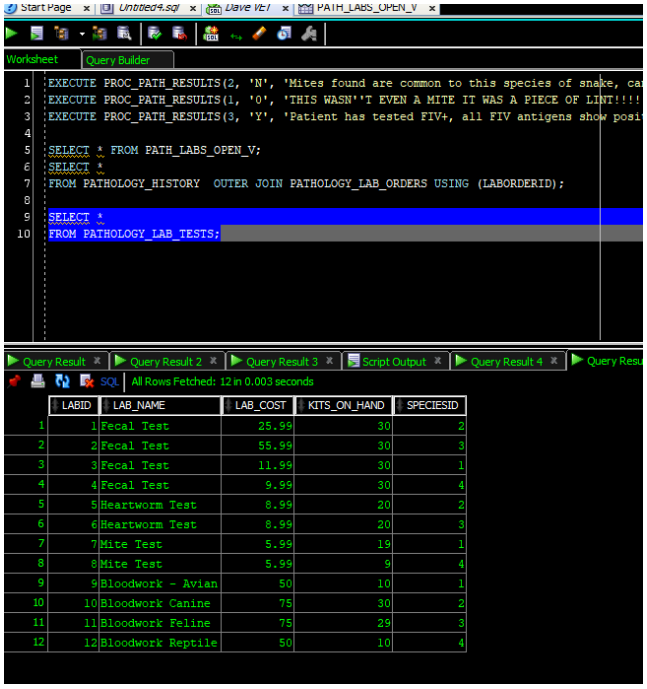


Figure 33

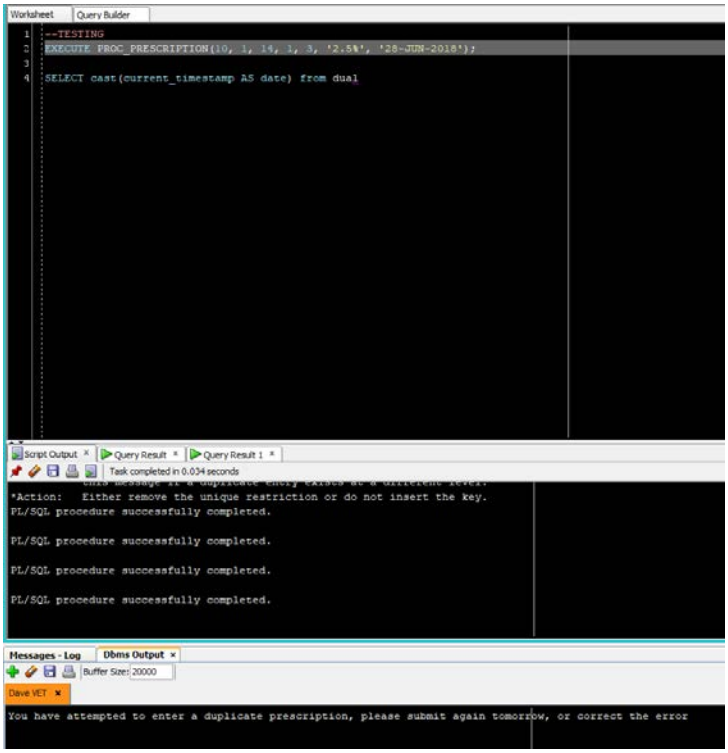


Figure 34

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.

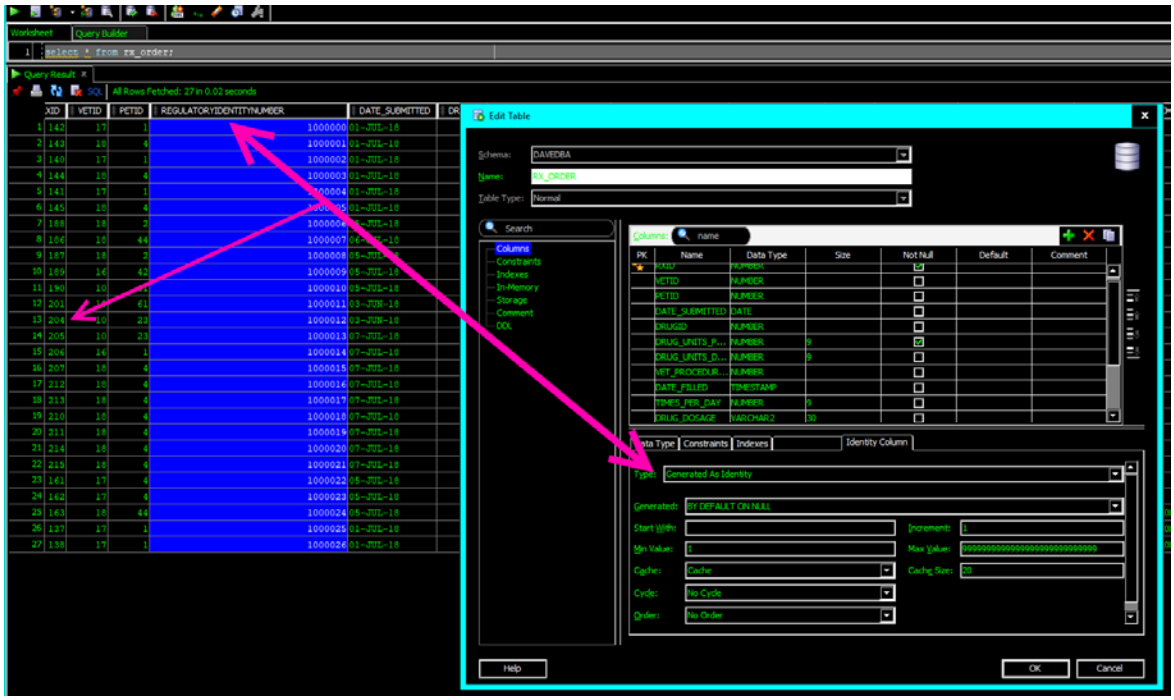


Figure 35

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] 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.

[7] Oracle's PL/SQL does have a Boolean data type, but that is not helpful during table creation.

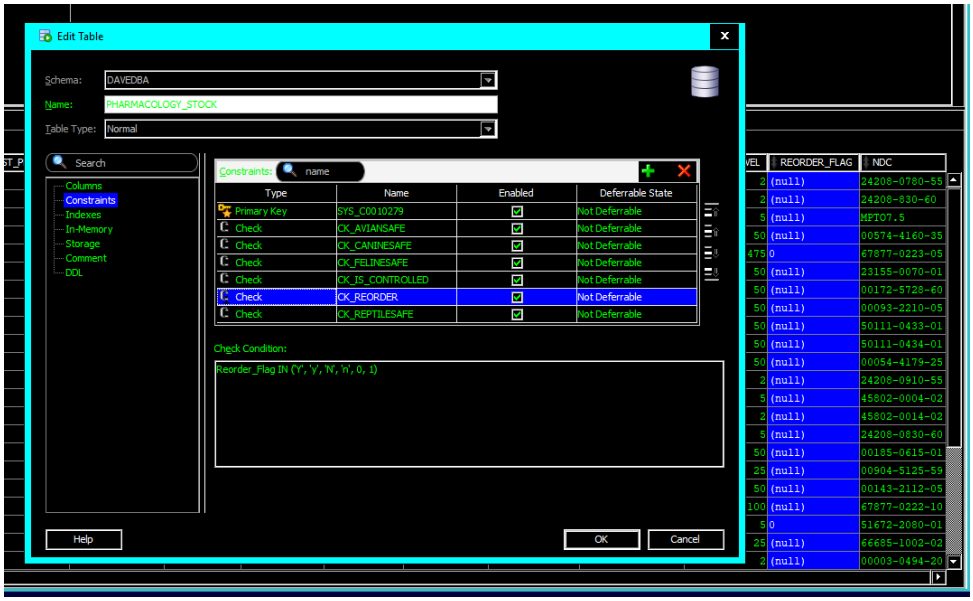


Figure 36

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].

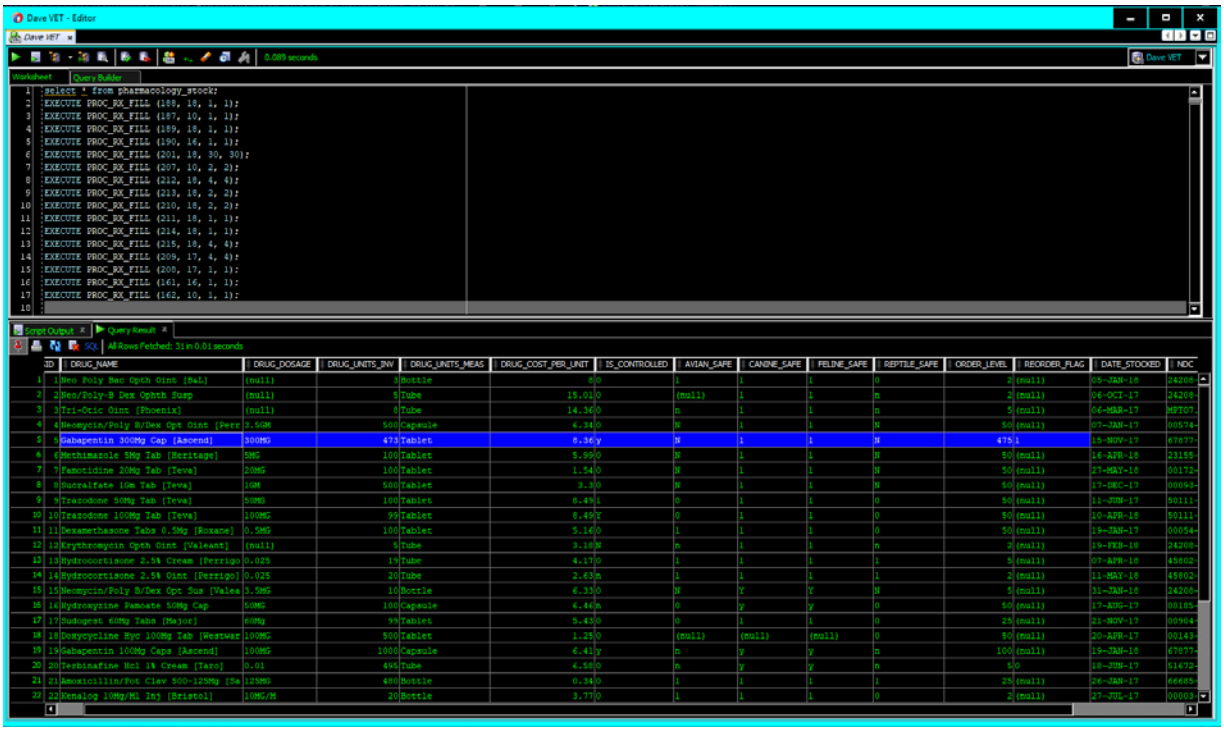


Figure 37

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.

Figure 38

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

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

CHART-04 SHOW ALL MEDICINES TAKEN BY PET EASILY

A view [RX_DETAILS_V] was created to make accessing patient data related to current medication much easier. Functions were used to translate drug, patient, and prescribing veterinarian name into human readable values; if the medication was given during a clinical procedure instead of a prescription, the procedural id was also translated into readable information [Figure 39].

Figure 39

An additional view was created only showing only maintenance medications for the animal. The view [RX_HISTORY5YRS_ALLMAINT_MEDS_V] was created to show any medication prescribed and marked as a maintenance medication within the last five years.

CHART-08 SEE ALL KNOWN MEDICAL PROCEDURES FOR THE ANIMAL

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

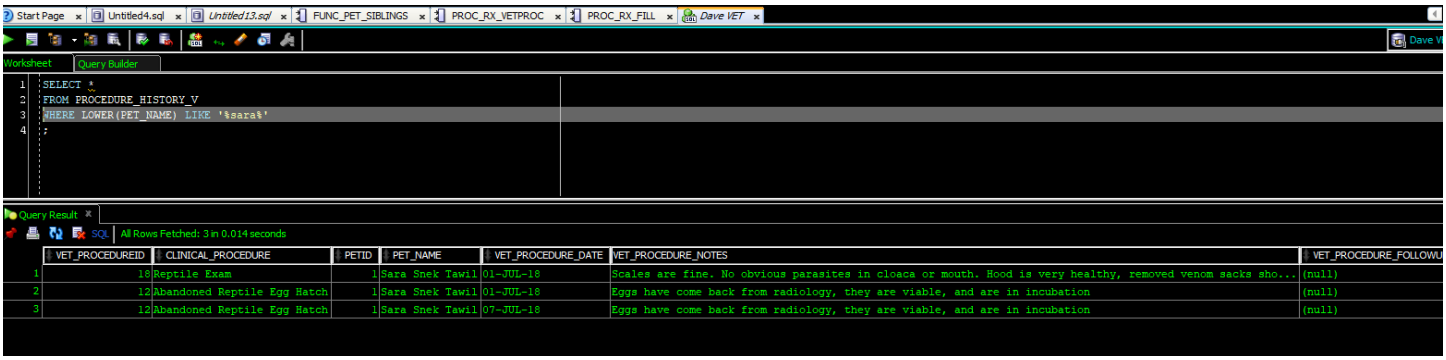


Figure 40

CHART–09 SEE THE MOST RECENT LAB-WORK DONE ON THE ANIMAL

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

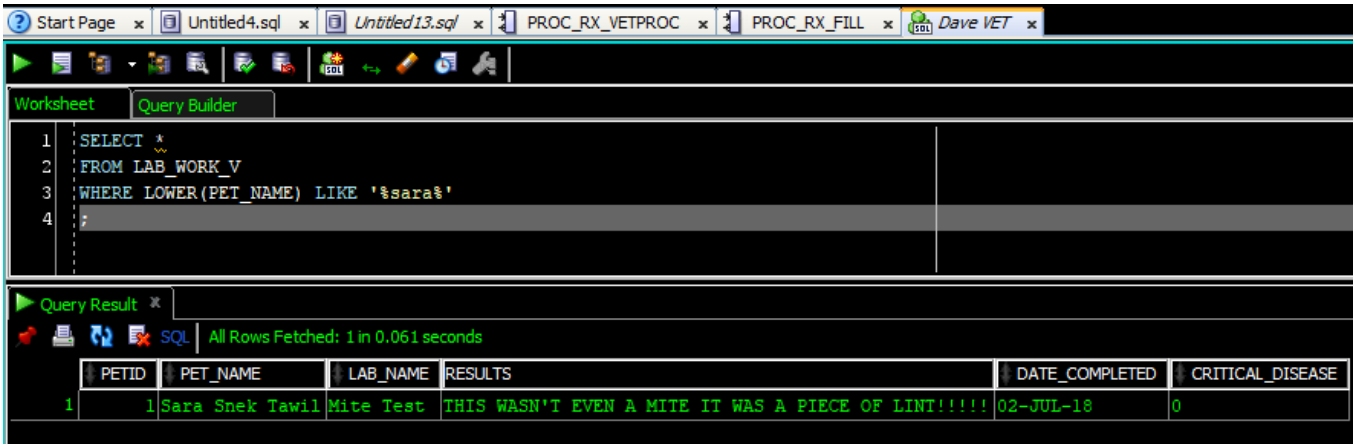


Figure 41

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] 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.

Figure 42

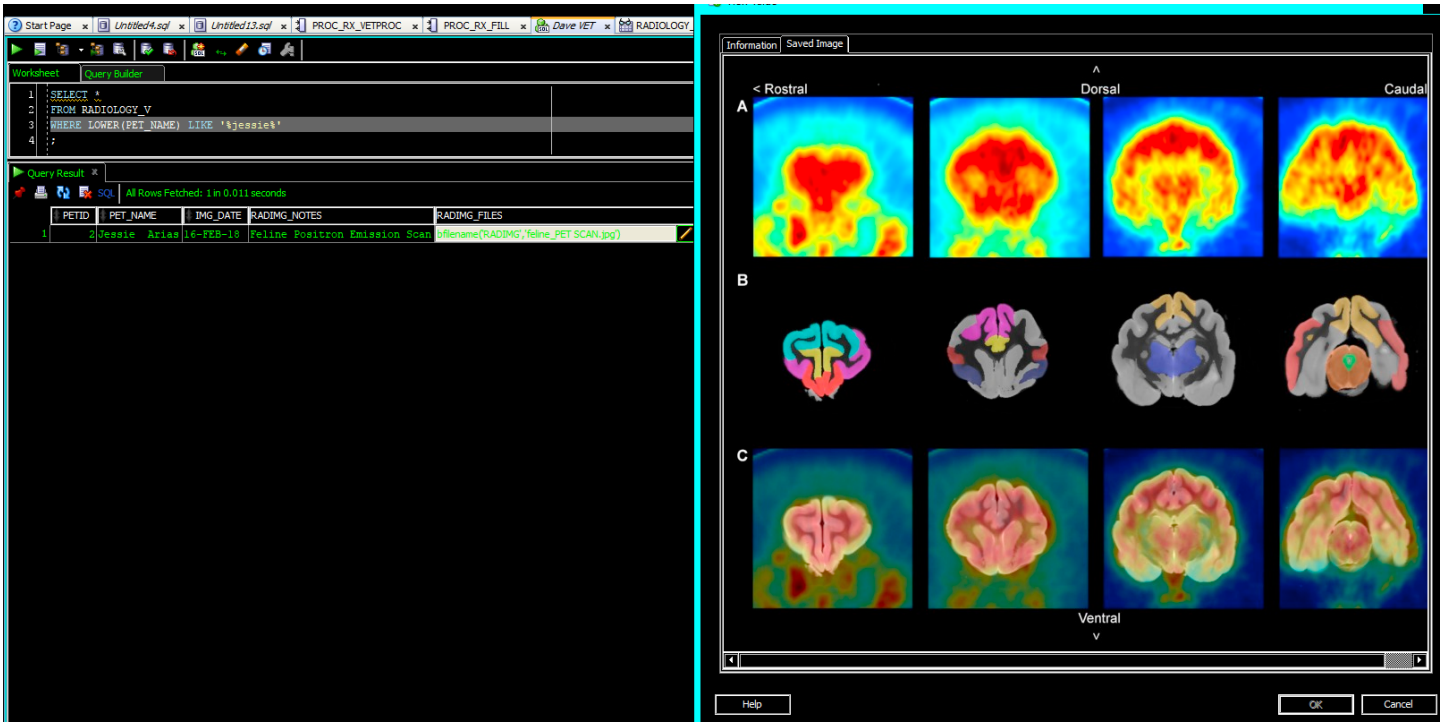


Figure 43

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.

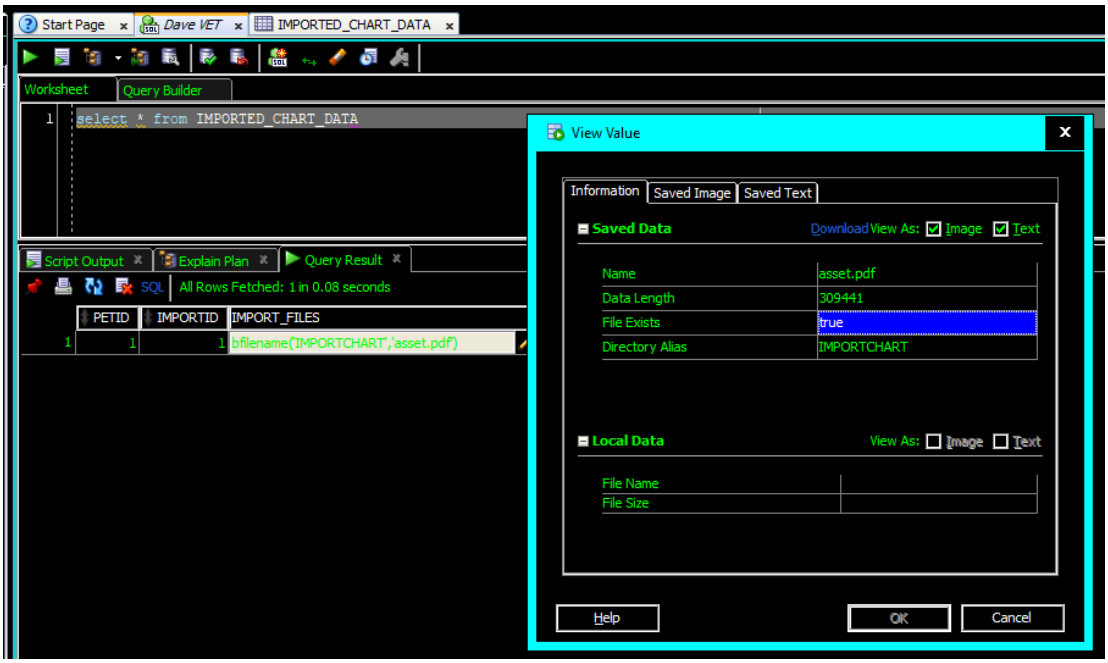


Figure 44

CHART-14 VET NEEDS TO SEE ANIMAL SIBLINGS OF THE PATIENT

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

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

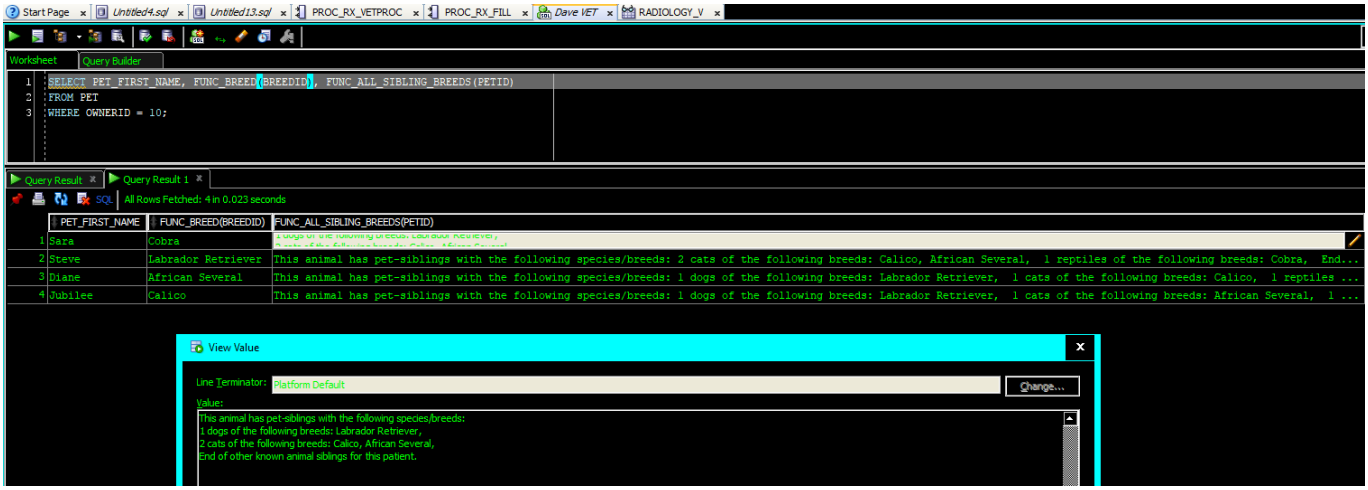


Figure 45

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.

Start Page | Untitled4.sql | Untitled13.sql | PROC_RX_VETPROC | PROC_RX_FILL | Dave VET | CHART_NOTES_V

Columns | Data | Grants | Dependencies | Details | Triggers | SQL

Sort.. Filter:

	PETID	VET	EVENT_DATE	EVENT	CRITDISEASE	NOTES	EVENT_TYPE
1	23	Emma Summers	(null)	Bloodwork Feline	(null)	(null)	PATHOLOGY
2	1	Eric Xavier	02-JUL-18	Mite Test	0	THIS WASN'T EVEN A MITE IT WAS A PIECE OF LINT!!!!!	PATHOLOGY
3	3	Ororo Lehnsherr	02-JUL-18	Mite Test	N	Mites found are common to this species of snake, can be removed with a g...	PATHOLOGY
4	2	Emma Summers	02-JUL-18	Bloodwork Feline	Y	Patient has tested FIV+, all FIV antigens show positive, patient is cont...	PATHOLOGY
5	1	Lucas Worthington	01-JUL-18	Reptile Exam	(null)	Scales are fine. No obvious parasites in cloaca or mouth. Hood is very h...	CLINICAL_PROCEDURE
6	1	Eric Xavier	01-JUL-18	Abandoned Reptile Egg Hatch	(null)	Eggs have come back from radiology, they are viable, and are in incubation	CLINICAL_PROCEDURE
7	4	Ororo Lehnsherr	01-JUL-18	TPLO Canine Left Hind Leg	(null)	Left Leg TPLO went well, he should be up and around in a couple of weeks	CLINICAL_PROCEDURE
8	2	Ororo Lehnsherr	05-JUL-18	Feline Exam	(null)	Found significant flea infection; fleas more aggressive than I've ever s...	CLINICAL_PROCEDURE
9	42	Eric Xavier	05-JUL-18	Abandoned Reptile Egg Hatch	(null)	Eggs abandoned by mom, put in hatchery, mom had some cloacal swelling ga...	CLINICAL_PROCEDURE
10	61	Emma Summers	05-JUL-18	Beak Scraping	(null)	Some beak swelling, possible infection, gave beak wash	CLINICAL_PROCEDURE
11	1	Eric Xavier	07-JUL-18	Abandoned Reptile Egg Hatch	(null)	Eggs have come back from radiology, they are viable, and are in incubation	CLINICAL_PROCEDURE
12	4	Ororo Lehnsherr	07-JUL-18	TPLO Canine Left Hind Leg	(null)	Left Leg TPLO went well, he should be up and around in a couple of weeks	CLINICAL_PROCEDURE
13	4	Ororo Lehnsherr	07-JUL-18	TPLO Canine Left Hind Leg	(null)	Left Leg TPLO went well, he should be up and around in a couple of weeks	CLINICAL_PROCEDURE
14	4	Ororo Lehnsherr	07-JUL-18	TPLO Canine Left Hind Leg	(null)	Left Leg TPLO went well, he should be up and around in a couple of weeks	CLINICAL_PROCEDURE
15	4	Lucas Worthington	05-JUL-18	Tumor Biopsy	(null)	Tumor biopsy taken of mass found on left foreleg, the procedure was unev...	CLINICAL_PROCEDURE
16	1	(null)	02-MAY-18	(null)	(null)	Not actually a monster it's a Cobra X-Ray	RADIOLOGY
17	4	(null)	01-JAN-18	(null)	(null)	Canine MRI	RADIOLOGY
18	2	(null)	16-FEB-18	(null)	(null)	Feline Positron Emission Scan	RADIOLOGY
19	42	(null)	14-NOV-17	(null)	(null)	Monitor Lizard CT scan	RADIOLOGY
20	1	Emma Summers	05-DEC-17	8	(null)	Snake is healthy; happy...for a snake.	ENCOUNTER
21	1	Lucas Worthington	02-JUL-18	8.5	(null)	Patient's parents was complaining of mites; it clearly is just lint.	ENCOUNTER
22	2	Eric Xavier	03-JUL-17	10	(null)	Post spaying checkup, cat is healthy and happy. No obvious pathologies....	ENCOUNTER
23	2	Ororo Lehnsherr	06-JUL-18	25	(null)	Patient has put on a significant amount of weight within one year! Putt...	ENCOUNTER
24	64	Staff Encounter	04-JUL-18	(null)	(null)	Owner alerted us of animal death on 03-MAR-08 Addtional notes including...	ENCOUNTER
25	46	Ororo Lehnsherr	04-JUL-18	(null)	(null)	In office death recorded notes including cause of death follow: Patient ...	ENCOUNTER

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]. A procedure was created to facilitate the opening of an estimate [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] is created.

Once an estimate number has been established [PROC_ESTIMATE_ADD] is used to add items to the estimate [Figure 47].

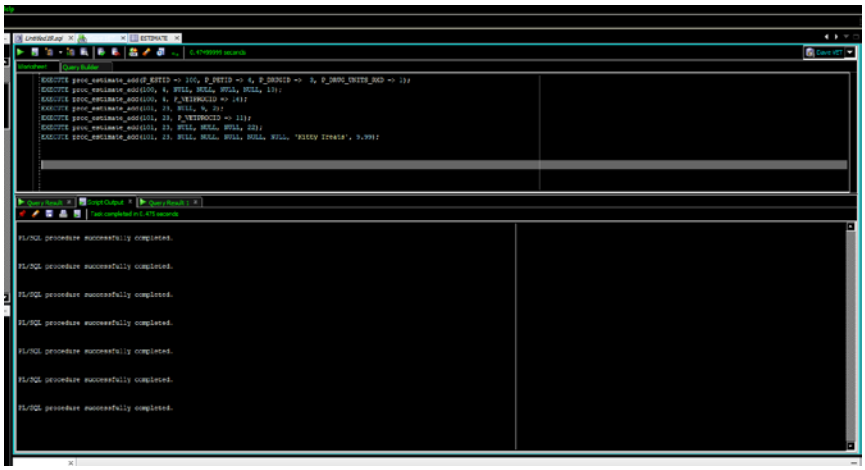
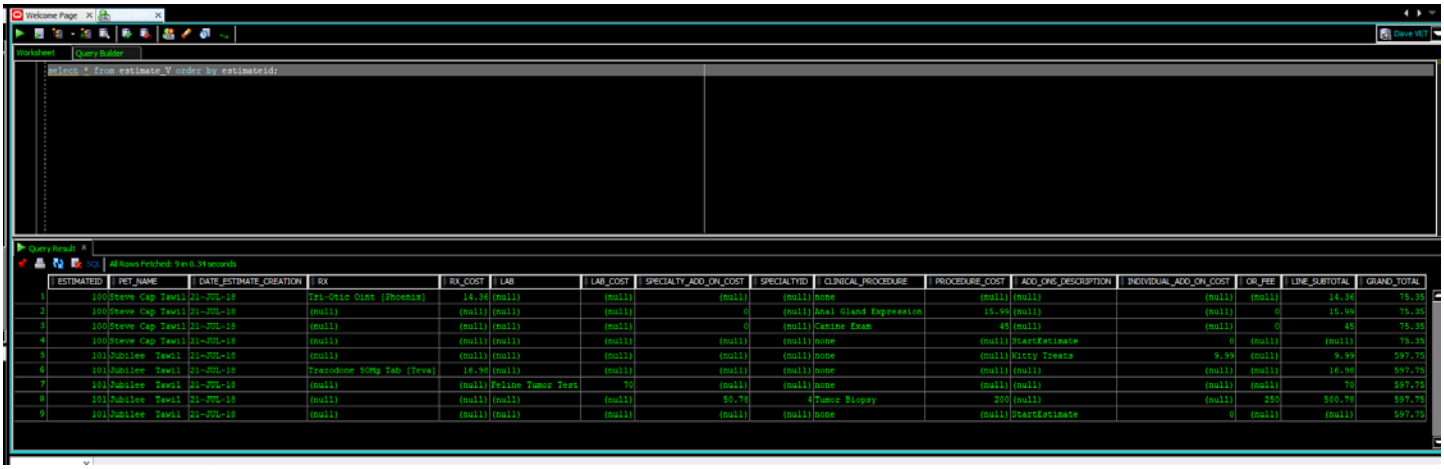


Figure 47

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



ESTIMATEID	PET_NAME	DATE_ESTIMATE_CREATION	RX	RX_COST	LAB	LAB_COST	SPECIALTY_ADD_ON_COST	SPECIALTYID	CLINICAL_PROCEDURE	PROCEDURE_COST	ADD_ONS_DESCRIPTION	INDIVIDUAL_ADD_ON_COST	OR_FEE	LINE_SUBTOTAL	GRAND_TOTAL
1	100	Steve Cap Tawil	21-JUL-18	Tri-Otic Oint (Phoenix)	14.36	(null)	(null)	(null)	(null)	none	(null)	(null)	(null)	(null)	14.36	75.35
2	100	Steve Cap Tawil	21-JUL-18	(null)	(null)	(null)	(null)	0	(null)	Anal Gland Expression	15.99	(null)	(null)	0	15.99	75.35
3	100	Steve Cap Tawil	21-JUL-18	(null)	(null)	(null)	(null)	0	(null)	Canine Exam	45	(null)	(null)	0	45	75.35
4	100	Steve Cap Tawil	21-JUL-18	(null)	(null)	(null)	(null)	(null)	(null)	none	(null)	StartEstimate	0	(null)	(null)	75.35
5	101	Dubilee Tawil	21-JUL-18	(null)	(null)	(null)	(null)	(null)	(null)	none	(null)	Kitty Treats	9.99	(null)	9.99	597.75
6	101	Dubilee Tawil	21-JUL-18	Trazodone 50Mg Tab (Teva)	16.98	(null)	(null)	(null)	(null)	none	(null)	(null)	(null)	(null)	16.98	597.75
7	101	Dubilee Tawil	21-JUL-18	(null)	(null)	Feline Tumor Test	70	(null)	(null)	none	(null)	(null)	(null)	(null)	70	597.75
8	101	Dubilee Tawil	21-JUL-18	(null)	(null)	(null)	(null)	50.78	4	Tumor Biopsy	200	(null)	(null)	250	500.78	597.75
9	101	Dubilee Tawil	21-JUL-18	(null)	(null)	(null)	(null)	(null)	(null)	none	(null)	StartEstimate	0	(null)	(null)	597.75

Figure 48

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.

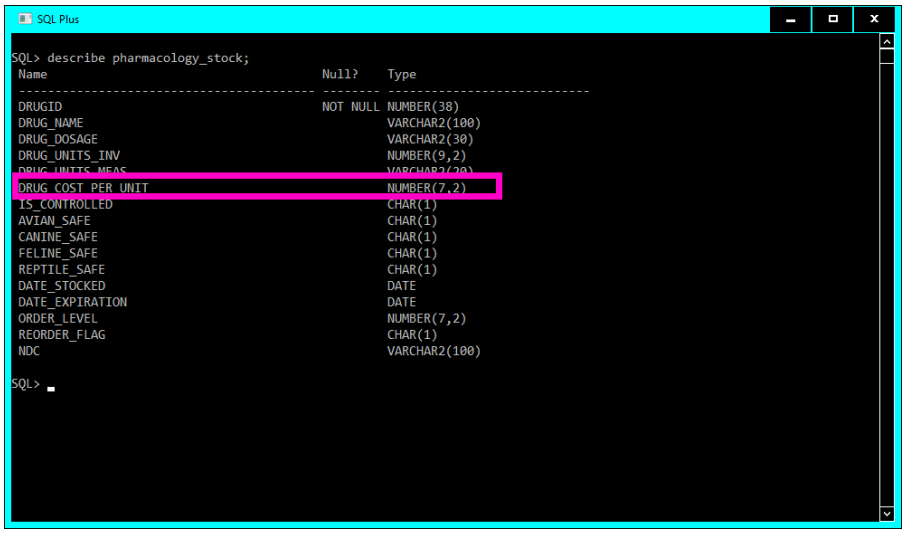


Figure 49

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] table. A view was created for invoices that considered INV-02's rule requirements regarding thirtyday grace periods before late fees [CUSTOMER_INV_V].

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_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**, note: the updated versions have Oracle style commenting in their header denoting the /*updated for invoicing support*/.

Figure 50

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.

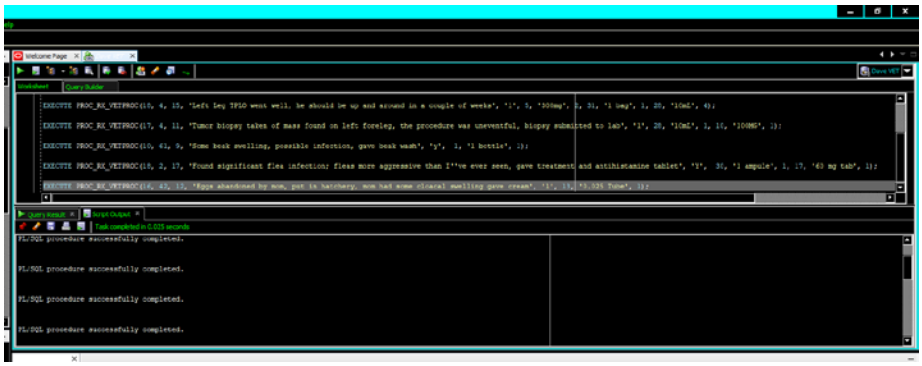


Figure 51

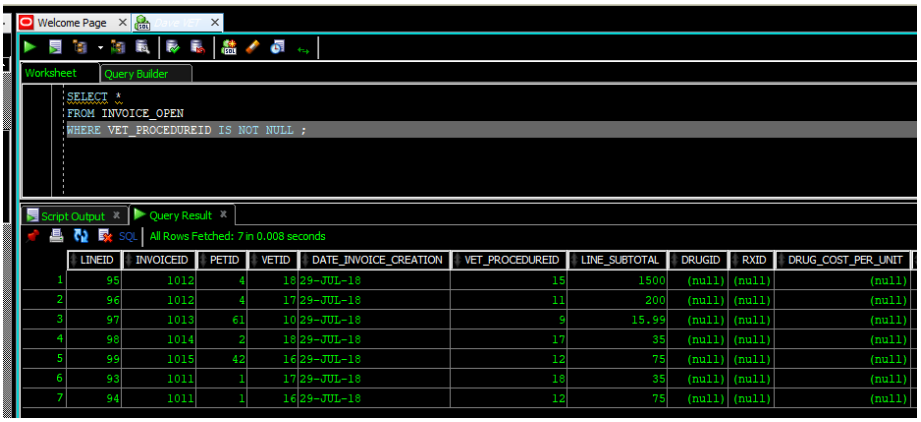


Figure 52

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

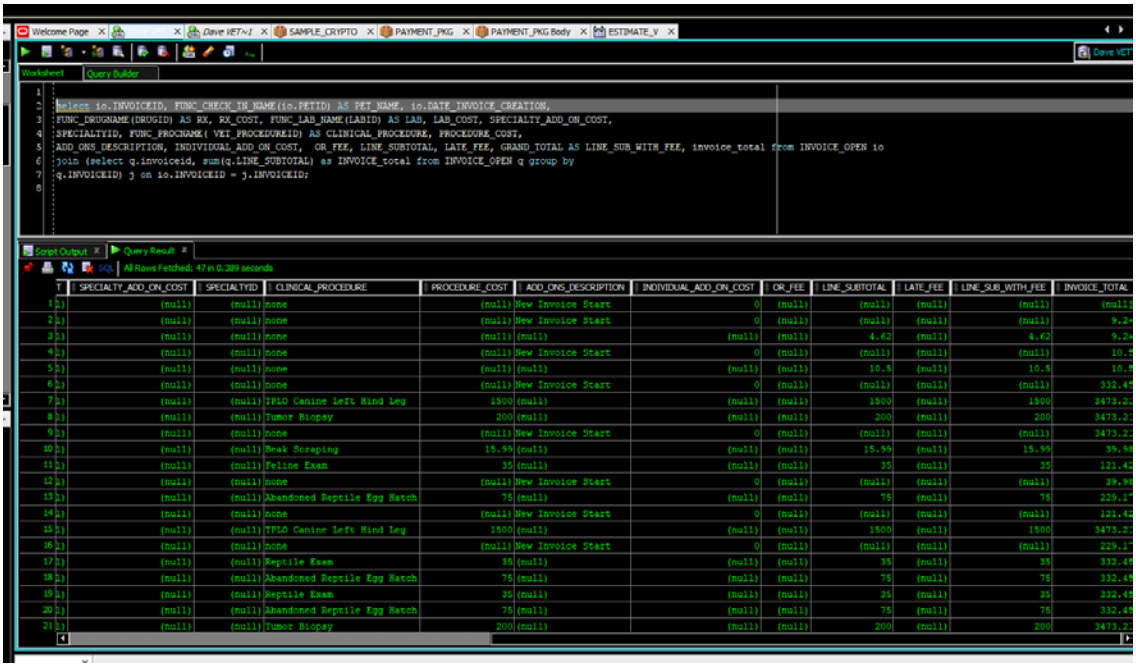


Figure 53

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], and one to add late fees [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 (temporally activated programs) were then created for the procedures [Figure 54].

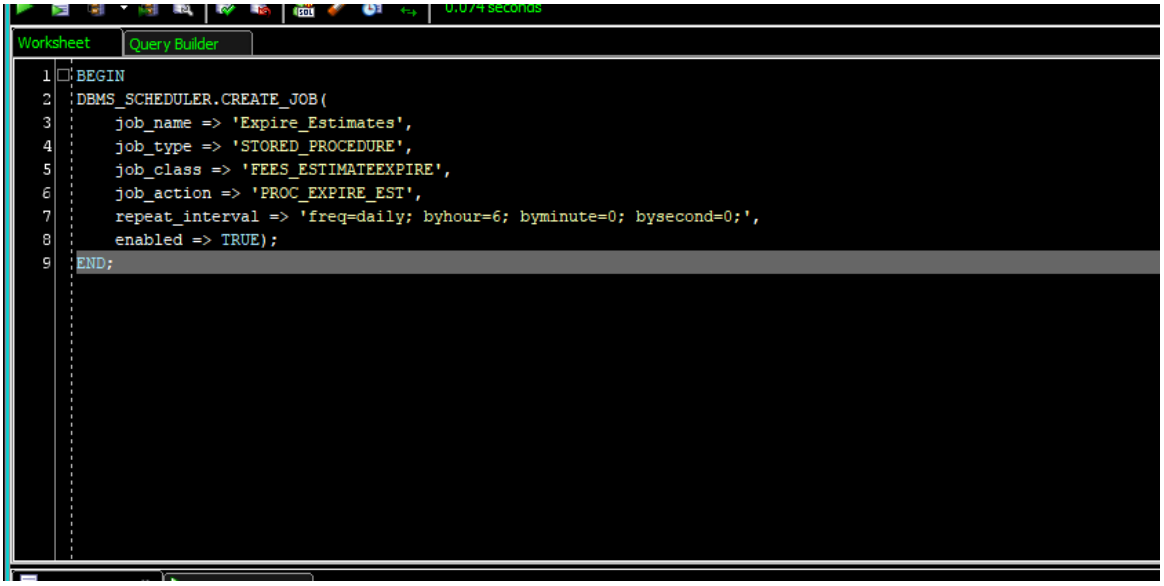


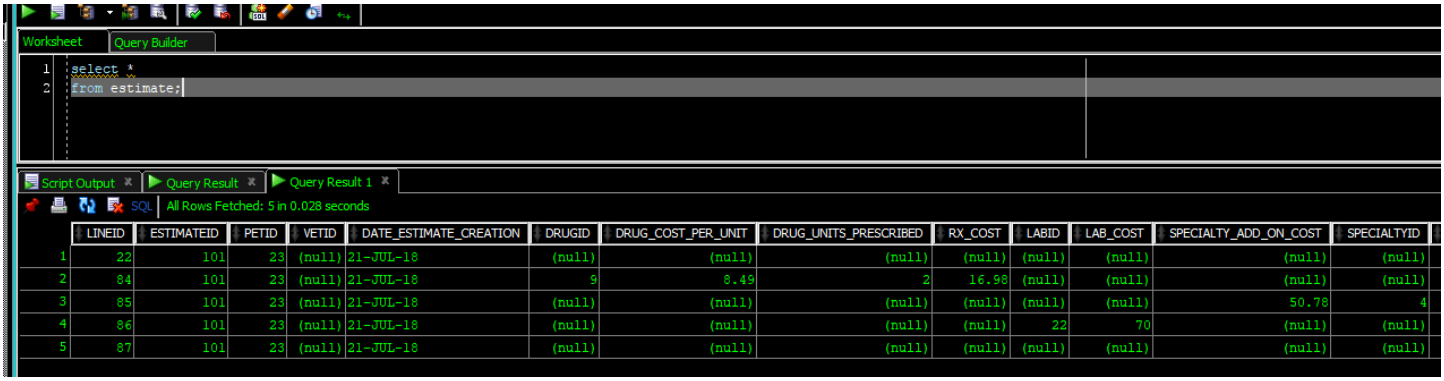
Figure 54

Figure 55 shows the estimate table pre-job run and Figure 56 shows the expired estimates gone.

Script Output | Query Result | All Rows Fetched: 9 in 0.019 seconds

	LINEID	ESTIMATEID	PETID	VETID	DATE_ESTIMATE_CREATION	DRUGID	DRUG_COST_PER_UNIT	DRUG_UNITS_PRESCRIBED	RX_COST	LABID	LAB_COST	SPECIALTY_ADD_ON_COST	SPECIALTYID	VET_PROCEDUREID	PROCEDURE_COST	ADD_ONS_DESC
1	21	100	4	(null)	05-MAY-18	(null)	(null)	(null)	(null)	(null)	(null)	(null)	(null)	(null)	(null)	StartEstimate
2	22	101	23	(null)	21-JUL-18	(null)	(null)	(null)	(null)	(null)	(null)	(null)	(null)	(null)	(null)	StartEstimate
3	81	100	4	(null)	05-MAY-18	3	14.36	1	14.36	(null)	(null)	(null)	(null)	(null)	(null)	(null)
4	82	100	4	(null)	05-MAY-18	(null)	(null)	(null)	(null)	(null)	(null)	0	(null)	10	15.99	(null)
5	83	100	4	(null)	05-MAY-18	(null)	(null)	(null)	(null)	(null)	(null)	0	(null)	16	45	(null)
6	84	101	23	(null)	21-JUL-18	9	8.49	2	16.98	(null)	(null)	(null)	(null)	(null)	(null)	(null)
7	85	101	23	(null)	21-JUL-18	(null)	(null)	(null)	(null)	(null)	(null)	50.78	4	11	200	(null)
8	86	101	23	(null)	21-JUL-18	(null)	(null)	(null)	(null)	22	70	(null)	(null)	(null)	(null)	(null)
9	87	101	23	(null)	21-JUL-18	(null)	(null)	(null)	(null)	(null)	(null)	(null)	(null)	(null)	(null)	Kitty Treats

Figure 55



```
select *
from estimate;
```

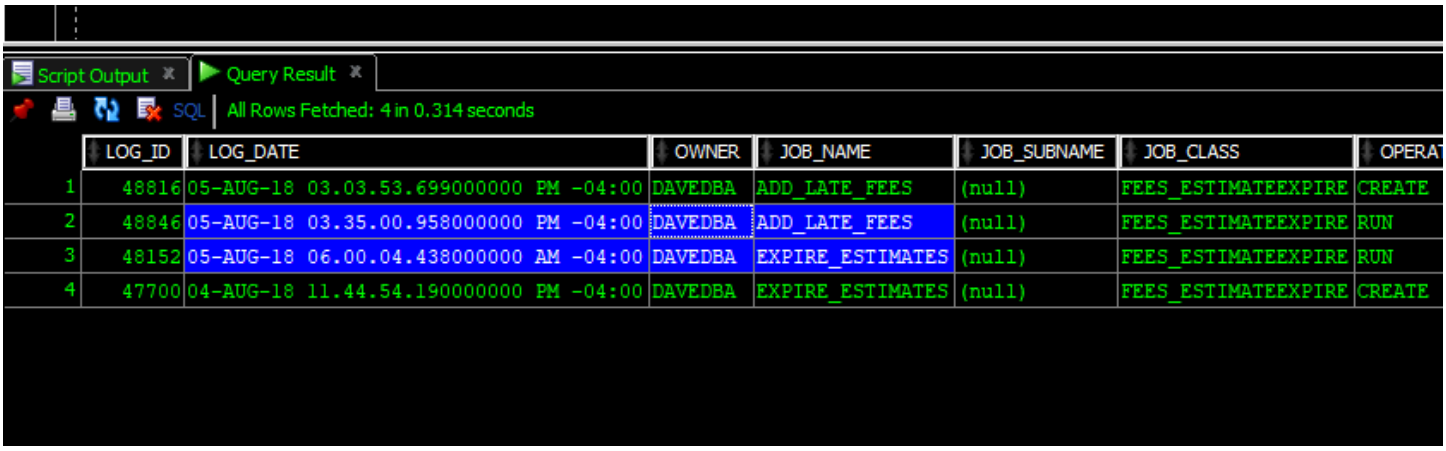
Script Output | Query Result | Query Result 1

All Rows Fetched: 5 in 0.028 seconds

	LINEID	ESTIMATEID	PETID	VETID	DATE_ESTIMATE_CREATION	DRUGID	DRUG_COST_PER_UNIT	DRUG_UNITS_PRESCRIBED	RX_COST	LABID	LAB_COST	SPECIALTY_ADD_ON_COST	SPECIALTYID
1	22	101	23	(null)	21-JUL-18	(null)	(null)	(null)	(null)	(null)	(null)	(null)	(null)
2	84	101	23	(null)	21-JUL-18	9	8.49	2	16.98	(null)	(null)	(null)	(null)
3	85	101	23	(null)	21-JUL-18	(null)	(null)	(null)	(null)	(null)	(null)	50.78	4
4	86	101	23	(null)	21-JUL-18	(null)	(null)	(null)	(null)	22	70	(null)	(null)
5	87	101	23	(null)	21-JUL-18	(null)	(null)	(null)	(null)	(null)	(null)	(null)	(null)

Figure 56

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].



Script Output | Query Result

All Rows Fetched: 4 in 0.314 seconds

	LOG_ID	LOG_DATE	OWNER	JOB_NAME	JOB_SUBNAME	JOB_CLASS	OPERA
1	48816	05-AUG-18 03.03.53.699000000 PM -04:00	DAVEDBA	ADD_LATE_FEES	(null)	FEES_ESTIMATEEXPIRE	CREATE
2	48846	05-AUG-18 03.35.00.958000000 PM -04:00	DAVEDBA	ADD_LATE_FEES	(null)	FEES_ESTIMATEEXPIRE	RUN
3	48152	05-AUG-18 06.00.04.438000000 AM -04:00	DAVEDBA	EXPIRE_ESTIMATES	(null)	FEES_ESTIMATEEXPIRE	RUN
4	47700	04-AUG-18 11.44.54.190000000 PM -04:00	DAVEDBA	EXPIRE_ESTIMATES	(null)	FEES_ESTIMATEEXPIRE	CREATE

Figure 57

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].

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].

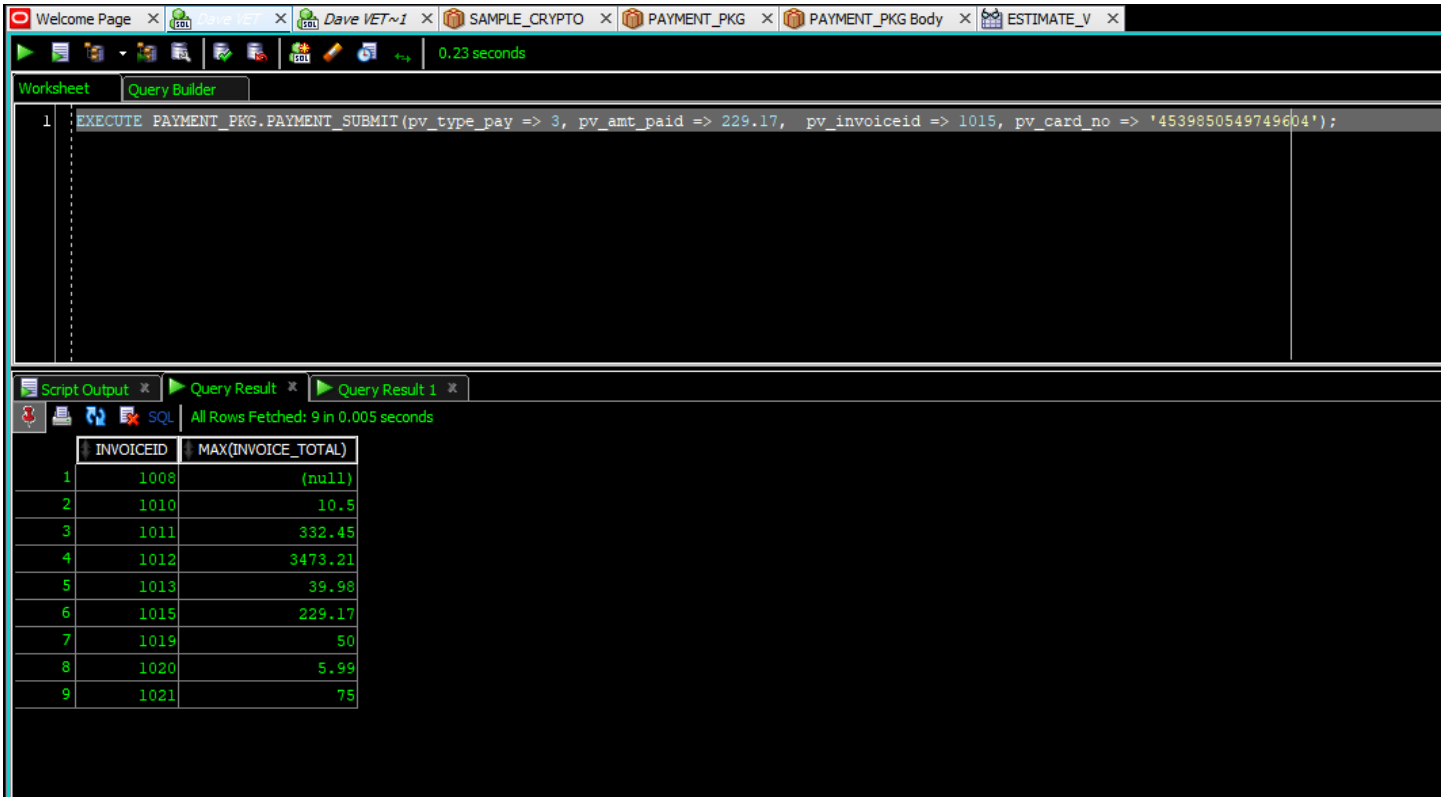


Figure 58

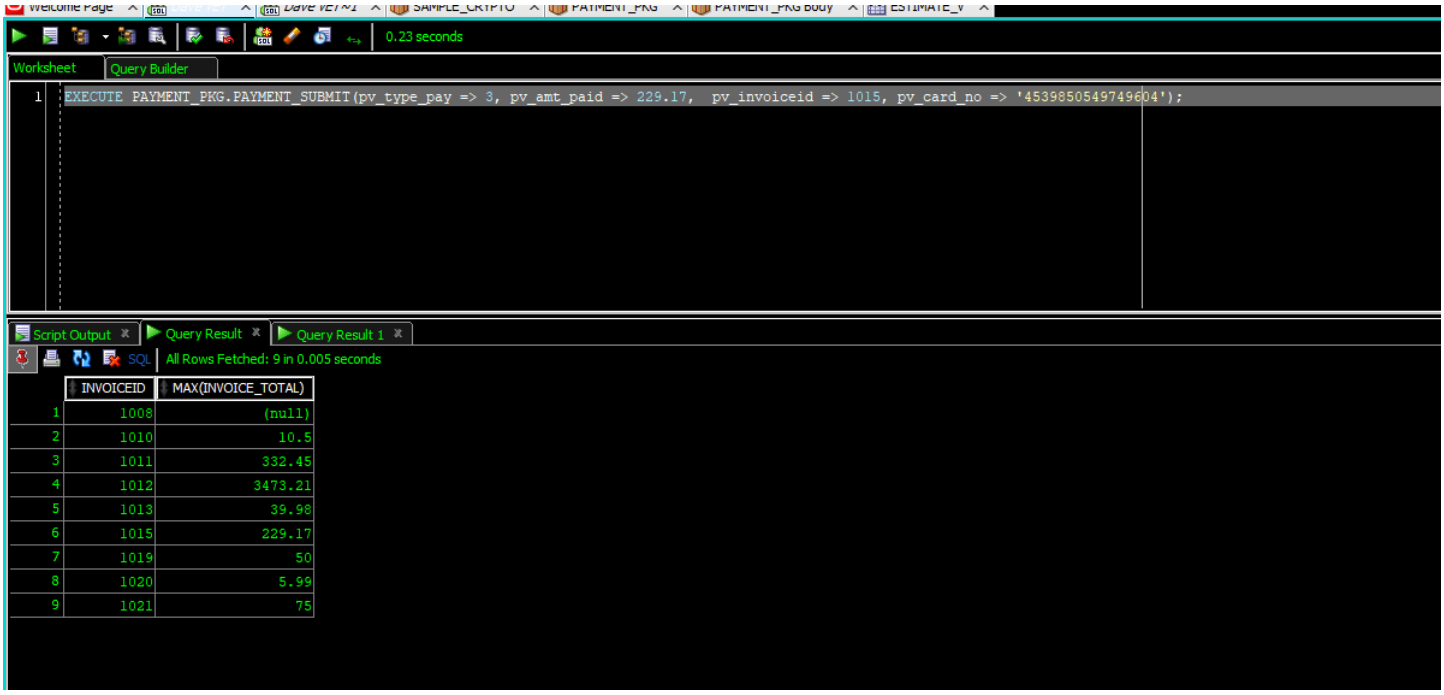


Figure 59

Figure 59 shows the removal of the invoice. This is accomplished by the subroutine INVOICE_MOVE of the [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] 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].

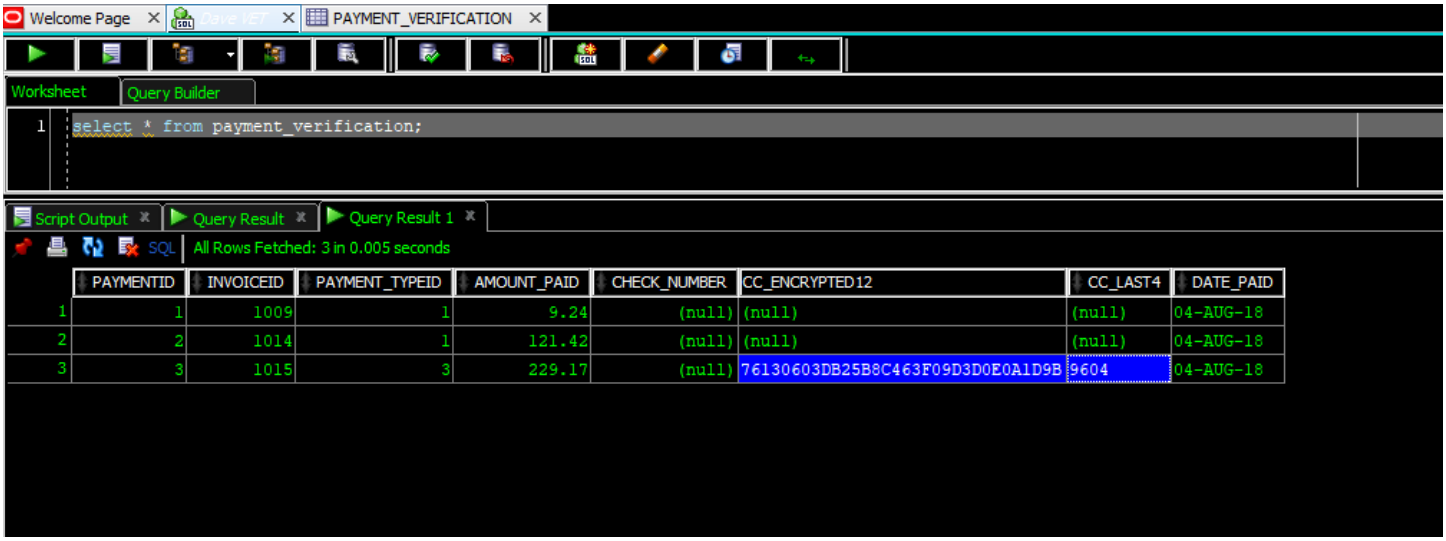


Figure 60

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] 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].

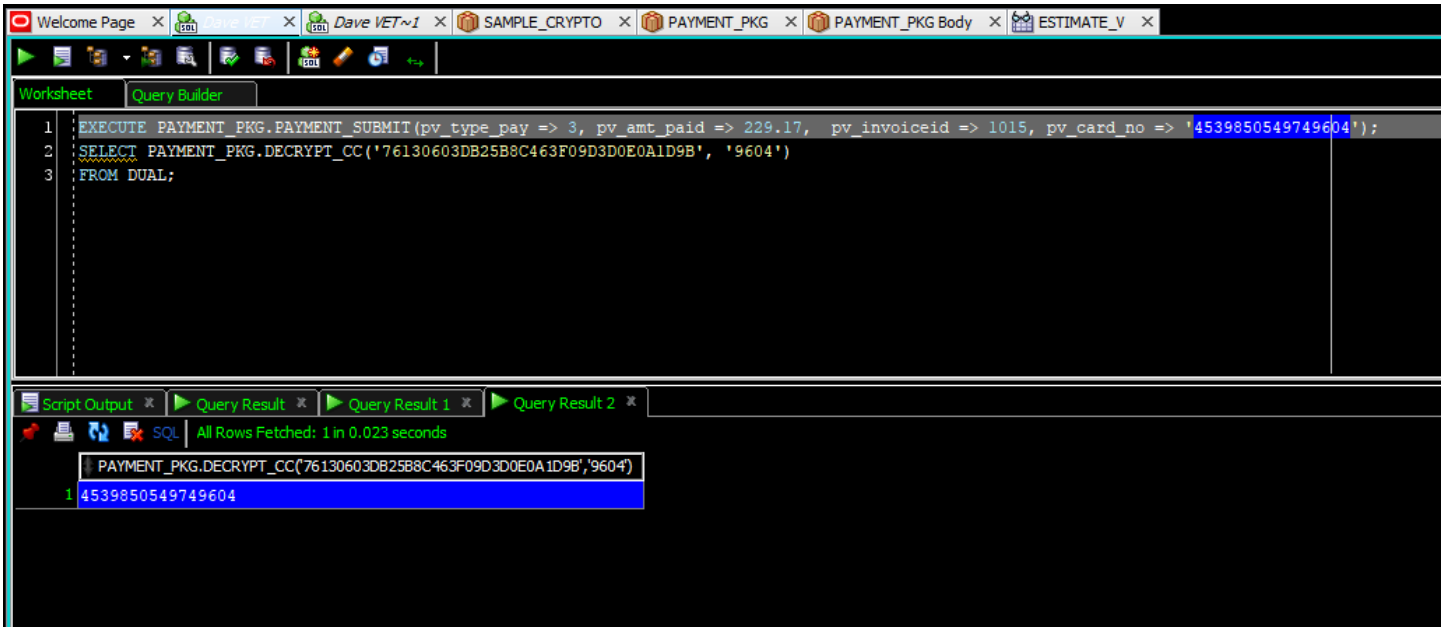


Figure 61

REPORTS REQUIRED

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

REPORT-04 MEDICINE SAFETY VS. SPECIES CHECK

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

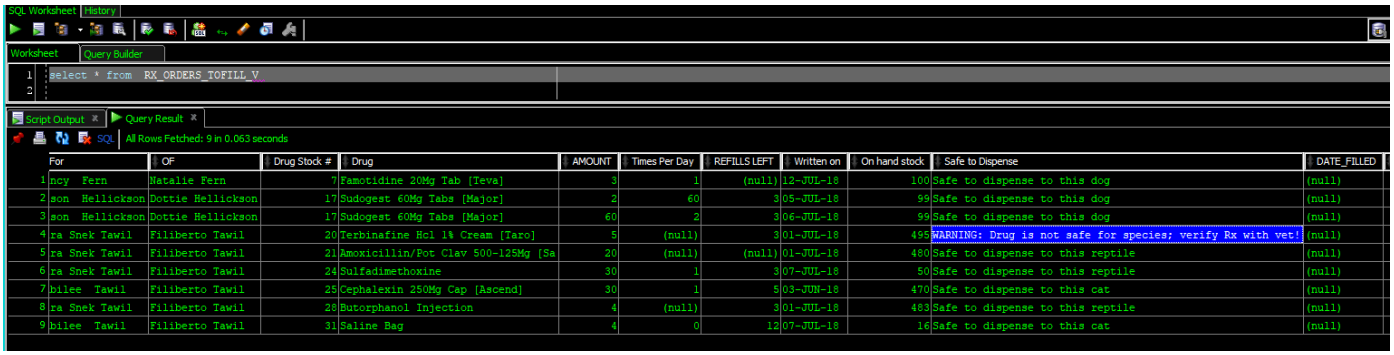
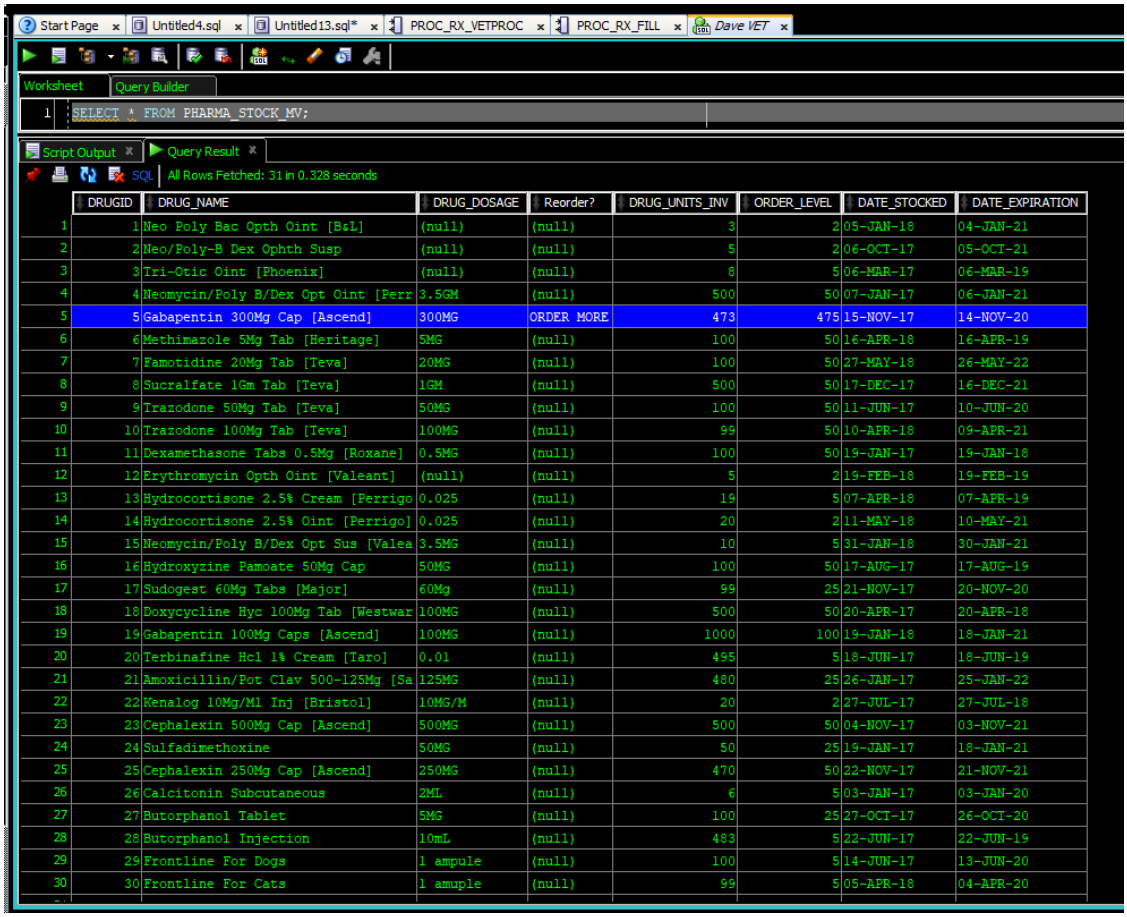


Figure 62

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].



Start Page | Untitled4.sql | Untitled13.sql* | PROC_RX_VETPROC | PROC_RX_FILL | Dave VET

Worksheet | Query Builder

```
SELECT * FROM PHARMA_STOCK_MV;
```

Script Output | Query Result

All Rows Fetched: 31 in 0.328 seconds

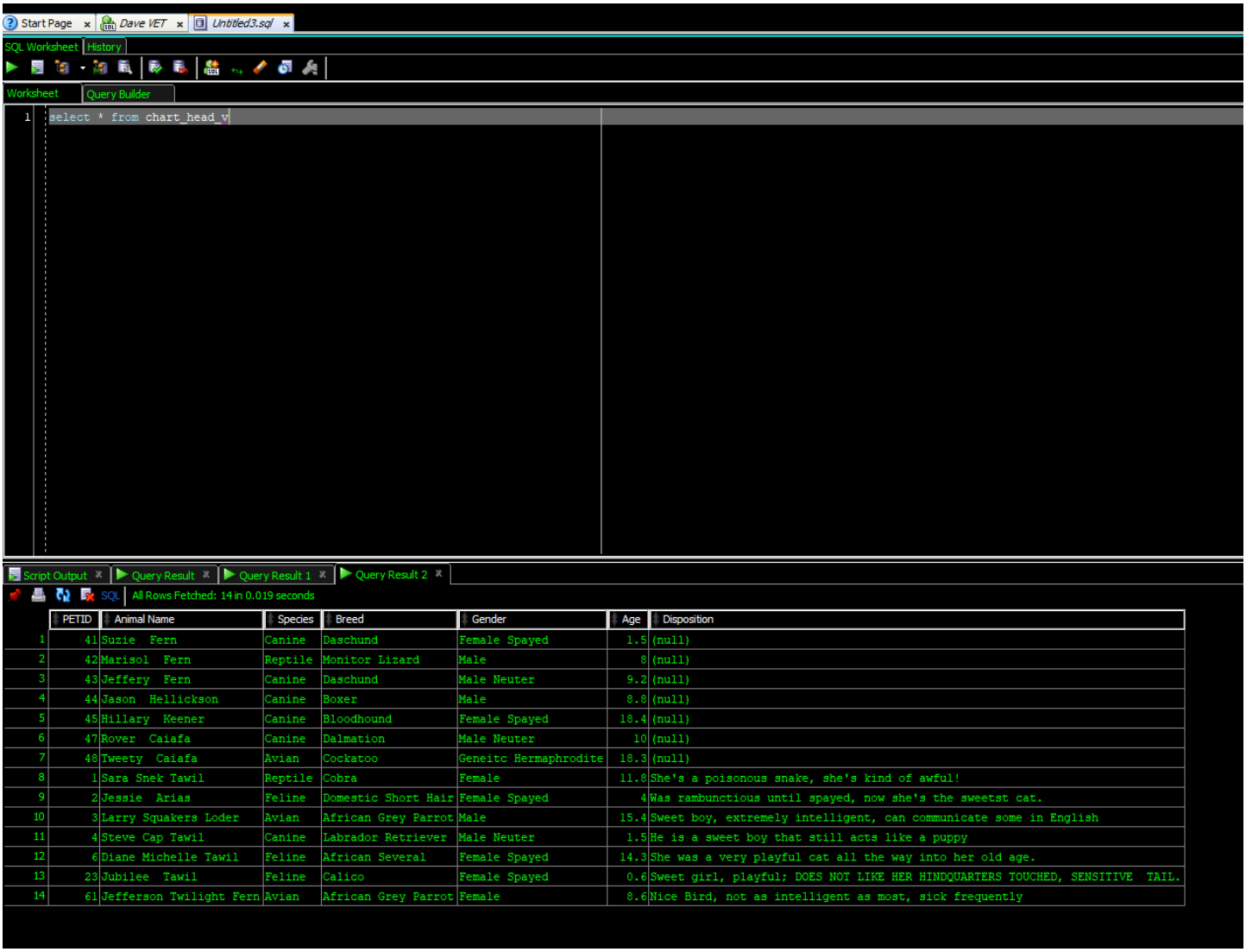
	DRUGID	DRUG_NAME	DRUG_DOSAGE	Reorder?	DRUG_UNITS_INV	ORDER_LEVEL	DATE_STOCKED	DATE_EXPIRATION
1	1	Neo Poly Bac Opth Oint [B&L]	(null)	(null)	3	2	05-JAN-18	04-JAN-21
2	2	Neo/Poly-B Dex Ophth Susp	(null)	(null)	5	2	06-OCT-17	05-OCT-21
3	3	Tri-Otic Oint [Phoenix]	(null)	(null)	8	5	06-MAR-17	06-MAR-19
4	4	Neomycin/Poly B/Dex Opt Oint [Perr	3.5GM	(null)	500	50	07-JAN-17	06-JAN-21
5	5	Gabapentin 300Mg Cap [Ascend]	300MG	ORDER MORE	473	475	15-NOV-17	14-NOV-20
6	6	Methimazole 5Mg Tab [Heritage]	5MG	(null)	100	50	16-APR-18	16-APR-19
7	7	Famotidine 20Mg Tab [Teva]	20MG	(null)	100	50	27-MAY-18	26-MAY-22
8	8	Sucralfate 1Gm Tab [Teva]	1GM	(null)	500	50	17-DEC-17	16-DEC-21
9	9	Trazodone 50Mg Tab [Teva]	50MG	(null)	100	50	11-JUN-17	10-JUN-20
10	10	Trazodone 100Mg Tab [Teva]	100MG	(null)	99	50	10-APR-18	09-APR-21
11	11	Dexamethasone Tabs 0.5Mg [Roxane]	0.5MG	(null)	100	50	19-JAN-17	19-JAN-18
12	12	Erythromycin Opth Oint [Valeant]	(null)	(null)	5	2	19-FEB-18	19-FEB-19
13	13	Hydrocortisone 2.5% Cream [Perrigo	0.025	(null)	19	5	07-APR-18	07-APR-19
14	14	Hydrocortisone 2.5% Oint [Perrigo]	0.025	(null)	20	2	11-MAY-18	10-MAY-21
15	15	Neomycin/Poly B/Dex Opt Sus [Valea	3.5MG	(null)	10	5	31-JAN-18	30-JAN-21
16	16	Hydroxyzine Pamoate 50Mg Cap	50MG	(null)	100	50	17-AUG-17	17-AUG-19
17	17	Sudogest 60Mg Tabs [Major]	60Mg	(null)	99	25	21-NOV-17	20-NOV-20
18	18	Doxycycline Hyc 100Mg Tab [Westwar	100MG	(null)	500	50	20-APR-17	20-APR-18
19	19	Gabapentin 100Mg Caps [Ascend]	100MG	(null)	1000	100	19-JAN-18	18-JAN-21
20	20	Terbinafine Hcl 1% Cream [Taro]	0.01	(null)	495	5	18-JUN-17	18-JUN-19
21	21	Amoxicillin/Pot Clav 500-125Mg [Sa	125MG	(null)	480	25	26-JAN-17	25-JAN-22
22	22	Kenalog 10Mg/Ml Inj [Bristol]	10MG/M	(null)	20	2	27-JUL-17	27-JUL-18
23	23	Cephalexin 500Mg Cap [Ascend]	500MG	(null)	500	50	04-NOV-17	03-NOV-21
24	24	Sulfadimethoxine	50MG	(null)	50	25	19-JAN-17	18-JAN-21
25	25	Cephalexin 250Mg Cap [Ascend]	250MG	(null)	470	50	22-NOV-17	21-NOV-21
26	26	Calcitonin Subcutaneous	2ML	(null)	6	5	03-JAN-17	03-JAN-20
27	27	Butorphanol Tablet	5MG	(null)	100	25	27-OCT-17	26-OCT-20
28	28	Butorphanol Injection	10mL	(null)	483	5	22-JUN-17	22-JUN-19
29	29	Frontline For Dogs	1 ampule	(null)	100	5	14-JUN-17	13-JUN-20
30	30	Frontline For Cats	1 amuple	(null)	99	5	05-APR-18	04-APR-20

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], this data contains the basic information that a vet would require immediately when walking into an examination room [Figure 49].



	PETID	Animal Name	Species	Breed	Gender	Age	Disposition
1	41	Suzie Fern	Canine	Daschund	Female Spayed	1.5	(null)
2	42	Marisol Fern	Reptile	Monitor Lizard	Male	8	(null)
3	43	Jeffery Fern	Canine	Daschund	Male Neuter	9.2	(null)
4	44	Jason Hellickson	Canine	Boxer	Male	8.8	(null)
5	45	Hillary Keener	Canine	Bloodhound	Female Spayed	18.4	(null)
6	47	Rover Caiafa	Canine	Dalmation	Male Neuter	10	(null)
7	48	Tweety Caiafa	Avian	Cockatoo	Geneitc Hermaphrodite	18.3	(null)
8	1	Sara Snek Tawil	Reptile	Cobra	Female	11.8	She's a poisonous snake, she's kind of awful!
9	2	Jessie Arias	Feline	Domestic Short Hair	Female Spayed	4	Was rambunctious until spayed, now she's the sweetst cat.
10	3	Larry Squakers Loder	Avian	African Grey Parrot	Male	15.4	Sweet boy, extremely intelligent, can communicate some in English
11	4	Steve Cap Tawil	Canine	Labrador Retriever	Male Neuter	1.5	He is a sweet boy that still acts like a puppy
12	6	Diane Michelle Tawil	Feline	African Several	Female Spayed	14.3	She was a very playful cat all the way into her old age.
13	23	Jubilee Tawil	Feline	Calico	Female Spayed	0.6	Sweet girl, playful; DOES NOT LIKE HER HINDQUARTERS TOUCHED, SENSITIVE TAIL.
14	61	Jefferson Twilight Fern	Avian	African Grey Parrot	Female	8.6	Nice Bird, not as intelligent as most, sick frequently

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].

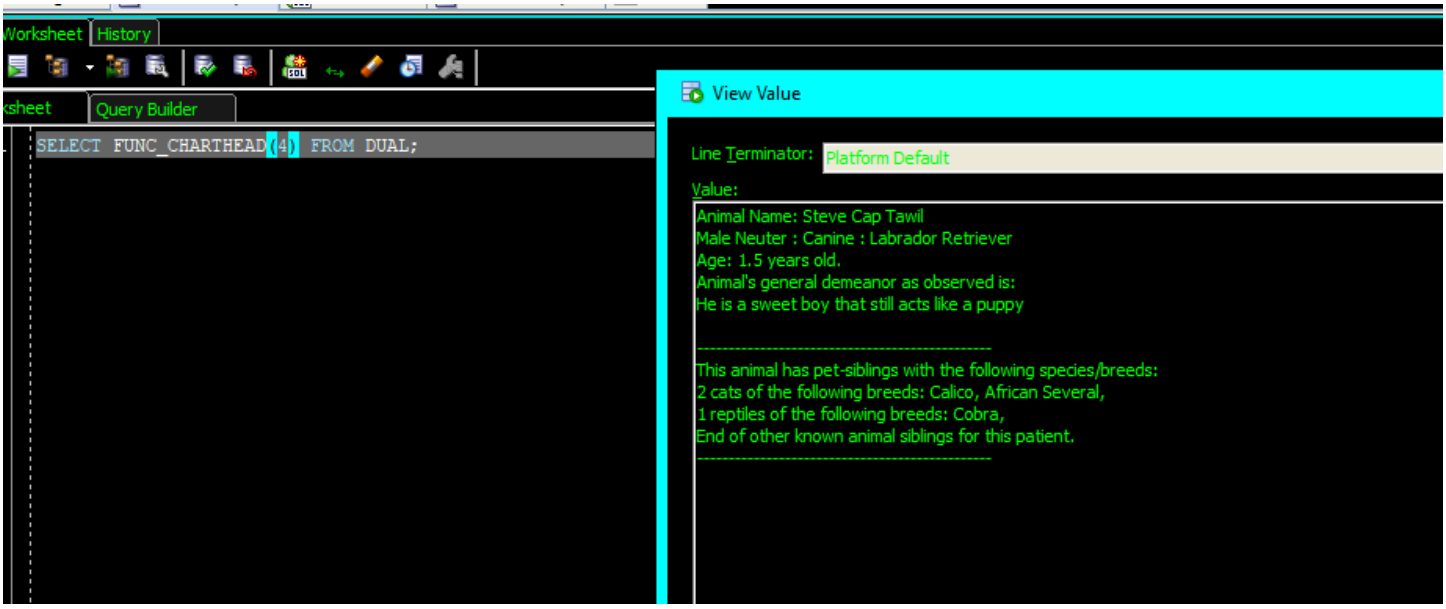


Figure 65

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

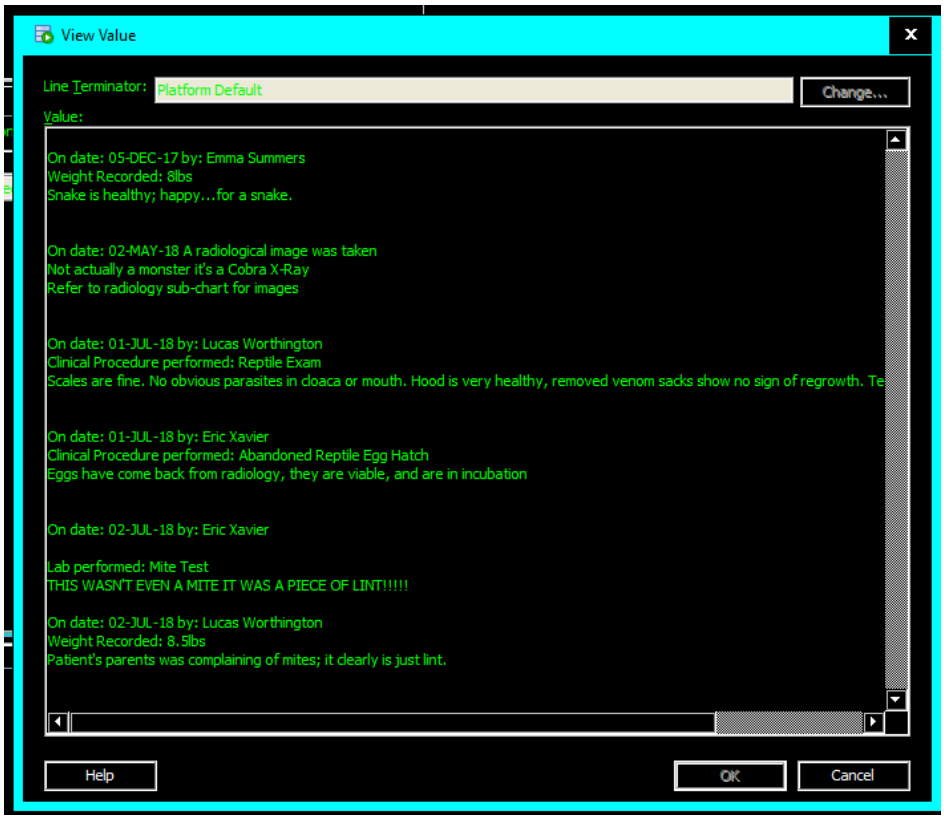


Figure 66

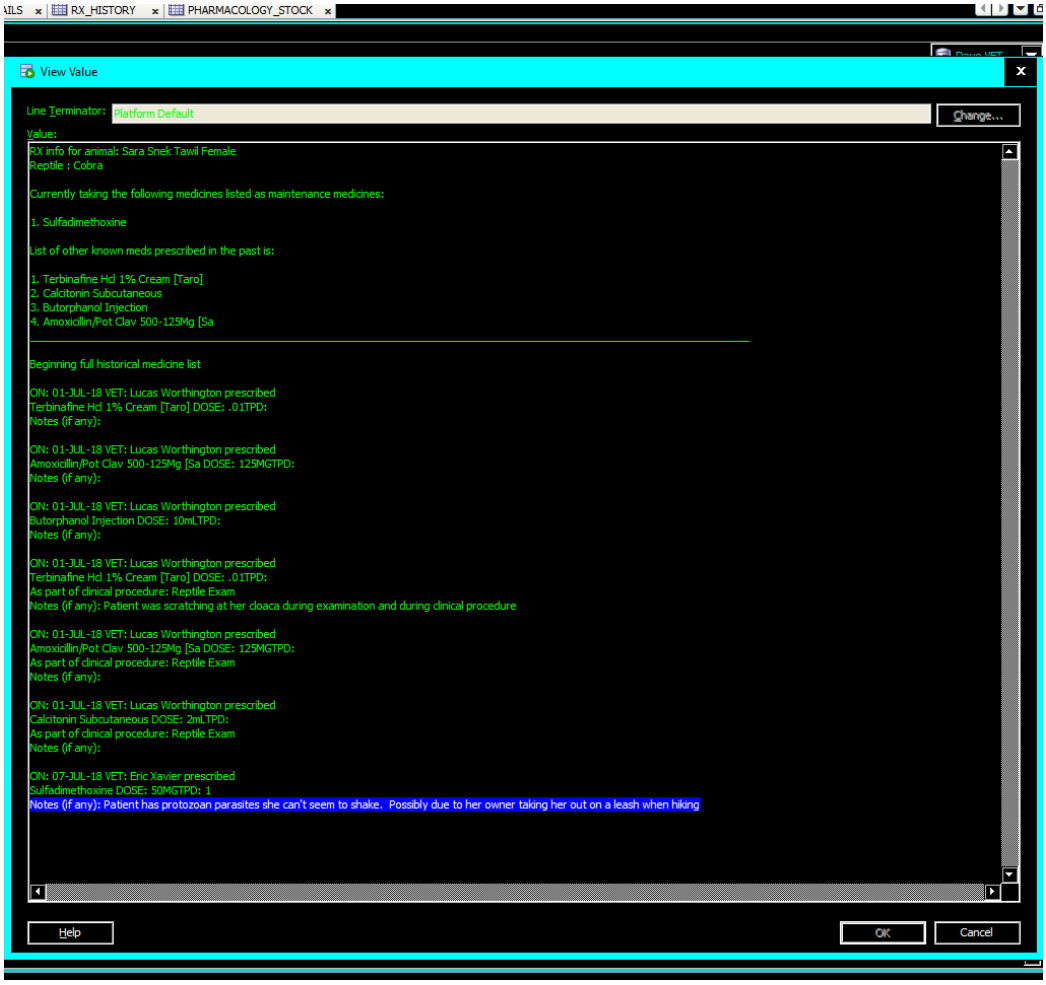


Figure 67

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

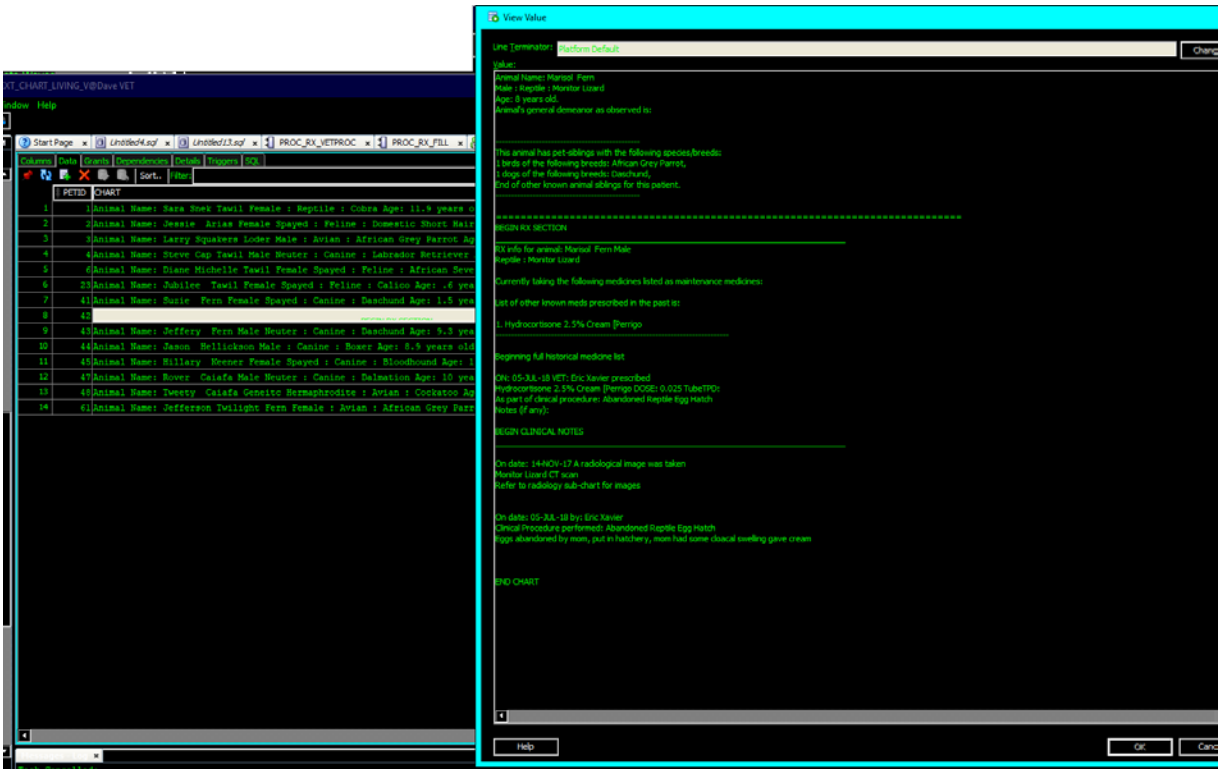


Figure 68

All these functions and procedures were wrapped in an Oracle Package [CHART_PKG].

REPORT-01: PART 2 TRADITIONALLY JOINED CHART

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

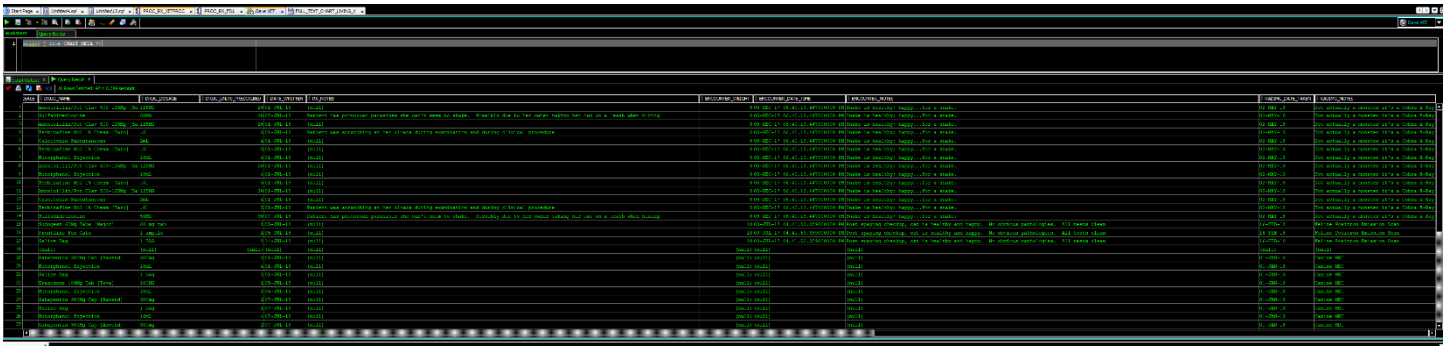


Figure 69

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] which feeds [CHART_META_V].

[8] Reminder: Right Click + Open Link from the popup menu will pop this document's images out in a browser window

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].

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] 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.

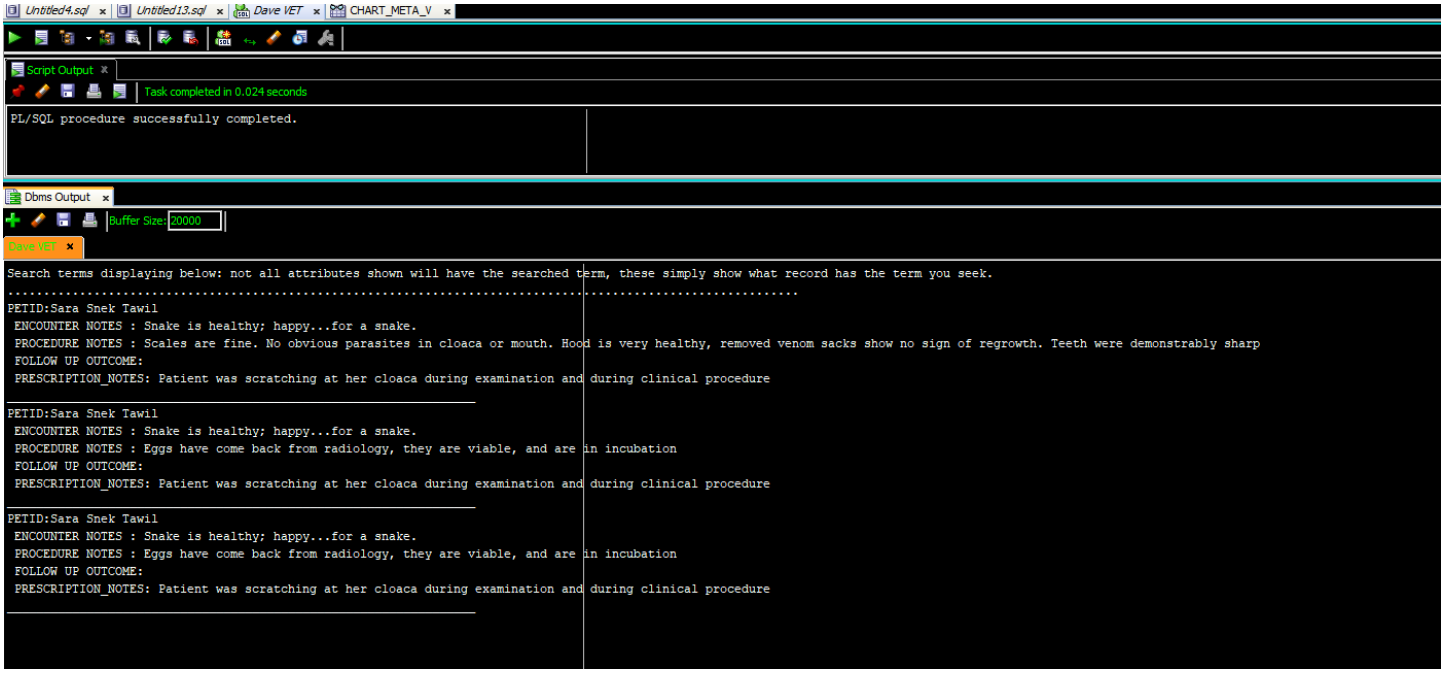


Figure 71

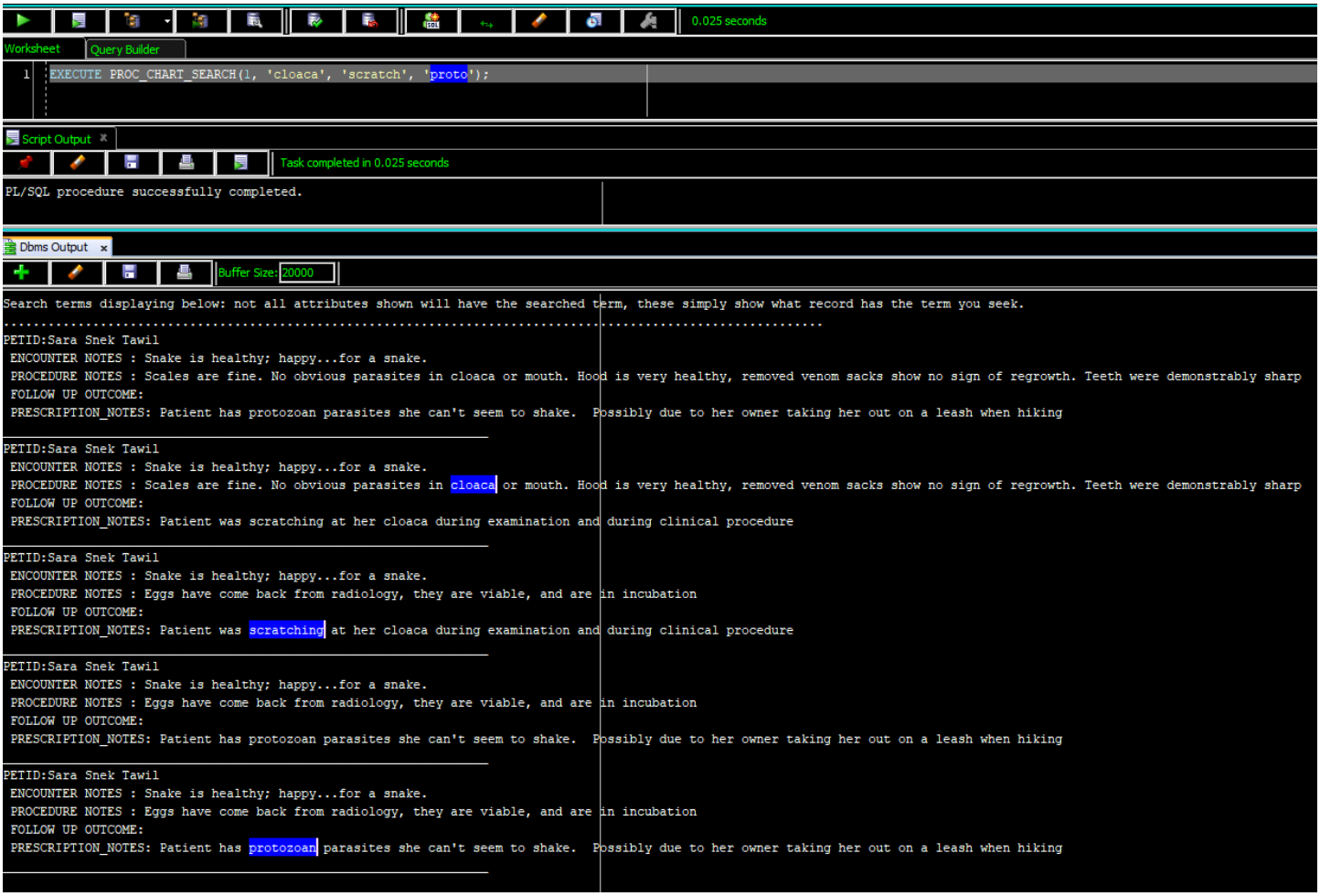


Figure 72

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].

BLOOD BANK

TRX-01 KEEP TRACK OF BLOOD INVENTORY

A table was created to satisfy this need [LOCAL_BLOOD_BANK].

TRX-11 SEE AVAILABLE BLOOD

Customer wanted to see how much blood they have on hand, and how much per species. To satisfy this need I was able to create a view using Oracle's powerful GROUP BY CUBE function as shown in Figure 58. A NVL function was created to fill in the top row as "Grand Total" instead of (null) to avoid confusion.

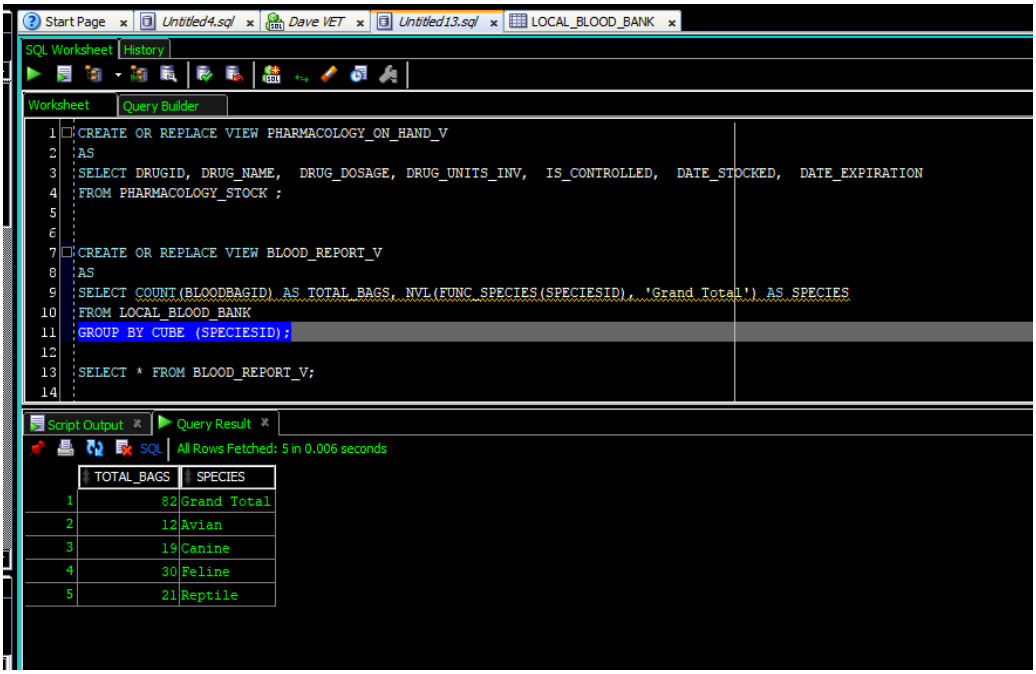


Figure 73

NOT INCLUDED RULES

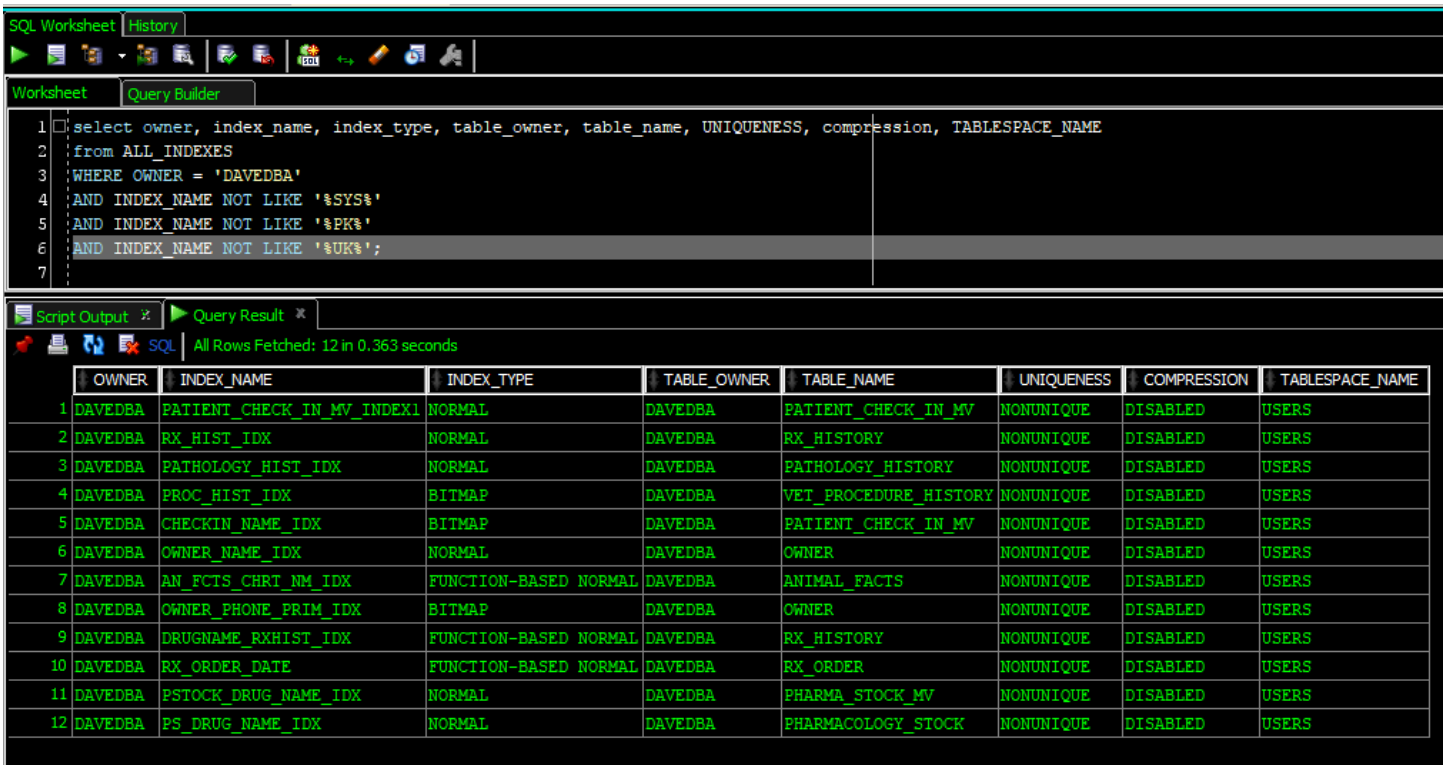
As with any project, requirements change or become obsolete during the building phase. The following anticipated business rules, transactions or reports were not included:

- 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.



	OWNER	INDEX_NAME	INDEX_TYPE	TABLE_OWNER	TABLE_NAME	UNIQUENESS	COMPRESSION	TABLESPACE_NAME
1	DAVEDBA	PATIENT_CHECK_IN_MV_INDEX1	NORMAL	DAVEDBA	PATIENT_CHECK_IN_MV	NONUNIQUE	DISABLED	USERS
2	DAVEDBA	RX_HIST_IDX	NORMAL	DAVEDBA	RX_HISTORY	NONUNIQUE	DISABLED	USERS
3	DAVEDBA	PATHOLOGY_HIST_IDX	NORMAL	DAVEDBA	PATHOLOGY_HISTORY	NONUNIQUE	DISABLED	USERS
4	DAVEDBA	PROC_HIST_IDX	BITMAP	DAVEDBA	VET_PROCEDURE_HISTORY	NONUNIQUE	DISABLED	USERS
5	DAVEDBA	CHECKIN_NAME_IDX	BITMAP	DAVEDBA	PATIENT_CHECK_IN_MV	NONUNIQUE	DISABLED	USERS
6	DAVEDBA	OWNER_NAME_IDX	NORMAL	DAVEDBA	OWNER	NONUNIQUE	DISABLED	USERS
7	DAVEDBA	AN_FCTS_CHRT_NM_IDX	FUNCTION-BASED NORMAL	DAVEDBA	ANIMAL_FACTS	NONUNIQUE	DISABLED	USERS
8	DAVEDBA	OWNER_PHONE_PRIM_IDX	BITMAP	DAVEDBA	OWNER	NONUNIQUE	DISABLED	USERS
9	DAVEDBA	DRUGNAME_RXHIST_IDX	FUNCTION-BASED NORMAL	DAVEDBA	RX_HISTORY	NONUNIQUE	DISABLED	USERS
10	DAVEDBA	RX_ORDER_DATE	FUNCTION-BASED NORMAL	DAVEDBA	RX_ORDER	NONUNIQUE	DISABLED	USERS
11	DAVEDBA	PSTOCK_DRUG_NAME_IDX	NORMAL	DAVEDBA	PHARMA_STOCK_MV	NONUNIQUE	DISABLED	USERS
12	DAVEDBA	PS_DRUG_NAME_IDX	NORMAL	DAVEDBA	PHARMACOLOGY_STOCK	NONUNIQUE	DISABLED	USERS

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.

```
1  SELECT * FROM DEPTREE ORDER BY SEQ#
```

Script Output | Query Result | SQL | All Rows Fetched: 29 in 1.29 seconds

	NESTED_LEVEL	TYPE	SCHEMA	NAME	SEQ#
1	0	TABLE	DAVEDBA	ANIMAL_FACTS	0
2	1	TRIGGER	DAVEDBA	PET_FACTS_TRIG	15
3	1	TRIGGER	DAVEDBA	PET_FACTS_UPDATE_TRIG	16
4	1	PROCEDURE	DAVEDBA	PROC_DEATH_RECORD	17
5	1	TRIGGER	DAVEDBA	GRIEF_ALERT_TRG	18
6	1	VIEW	DAVEDBA	CHART_HEAD_V	19
7	1	VIEW	DAVEDBA	FULL_TEXT_CHART_LIVING_V	20
8	1	VIEW	DAVEDBA	FULL_TEXT_CHART_DECEASED_V	21
9	1	MATERIALIZED VIEW	DAVEDBA	CHART_META_GROUPED_SETS_MV	22
10	1	PACKAGE	DAVEDBA	CHART_PKG	23
11	2	VIEW	DAVEDBA	CHART_HEAD_V	24
12	2	VIEW	DAVEDBA	RX_DETAILS_V	25
13	3	PACKAGE BODY	DAVEDBA	CHART_PKG	26
14	2	VIEW	DAVEDBA	FULL_TEXT_CHART_LIVING_V	27
15	2	VIEW	DAVEDBA	FULL_TEXT_CHART_DECEASED_V	28
16	2	VIEW	DAVEDBA	CHART_META_V	29
17	2	VIEW	DAVEDBA	CHART_META_GROUPED_SETS_V	30
18	2	VIEW	DAVEDBA	LAB_WORK_V	31
19	2	VIEW	DAVEDBA	PROCEDURE_HISTORY_V	32
20	2	PACKAGE BODY	DAVEDBA	CHART_PKG	33
21	2	VIEW	DAVEDBA	SPLIT_TEXT_CHART_LVNG_V	34
22	2	VIEW	DAVEDBA	RADIOLOGY_V	35
23	2	INDEX	DAVEDBA	AN_FCTS_CHRT_NM_IDX	36
24	2	PROCEDURE	DAVEDBA	PROC_CHART_SEARCH	37
25	1	PACKAGE BODY	DAVEDBA	CHART_PKG	38
26	1	VIEW	DAVEDBA	SPLIT_TEXT_CHART_LVNG_V	39
27	1	FUNCTION	DAVEDBA	FUNC_RX_SPECIES_SAFE	40
28	2	VIEW	DAVEDBA	RX_ORDERS_TOFILL_V	41
29	1	MATERIALIZED VIEW	DAVEDBA	CHART_META_MV	42

Figure 75

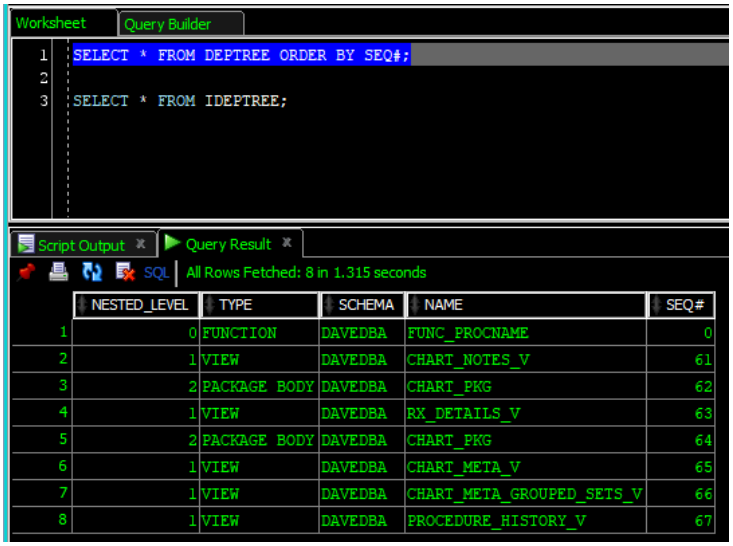
```
1  SELECT * FROM IDEPTREE;
```

Worksheet | Query Builder

Script Output | Query Result | SQL | All Rows Fetched: 15 in 2.407 seconds

	DEPENDENCIES
1	PACKAGE DAVEDBA.CHART_PKG
2	VIEW DAVEDBA.CHART_HEAD_V
3	VIEW DAVEDBA.RX_DETAILS_V
4	PACKAGE BODY DAVEDBA.CHART_PKG
5	VIEW DAVEDBA.FULL_TEXT_CHART_LIVING_V
6	VIEW DAVEDBA.FULL_TEXT_CHART_DECEASED_V
7	VIEW DAVEDBA.CHART_META_V
8	VIEW DAVEDBA.CHART_META_GROUPED_SETS_V
9	VIEW DAVEDBA.LAB_WORK_V
10	VIEW DAVEDBA.PROCEDURE_HISTORY_V
11	PACKAGE BODY DAVEDBA.CHART_PKG
12	VIEW DAVEDBA.SPLIT_TEXT_CHART_LVNG_V
13	VIEW DAVEDBA.RADIOLOGY_V
14	INDEX DAVEDBA.AN_FCTS_CHRT_NM_IDX
15	PROCEDURE DAVEDBA.PROC_CHART_SEARCH

Figure 76



	NESTED_LEVEL	TYPE	SCHEMA	NAME	SEQ#
1	0	FUNCTION	DAVEDBA	FUNC_PROCNAME	0
2	1	VIEW	DAVEDBA	CHART_NOTES_V	61
3	2	PACKAGE BODY	DAVEDBA	CHART_PKG	62
4	1	VIEW	DAVEDBA	RX_DETAILS_V	63
5	2	PACKAGE BODY	DAVEDBA	CHART_PKG	64
6	1	VIEW	DAVEDBA	CHART_META_V	65
7	1	VIEW	DAVEDBA	CHART_META_GROUPED_SETS_V	66
8	1	VIEW	DAVEDBA	PROCEDURE_HISTORY_V	67

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

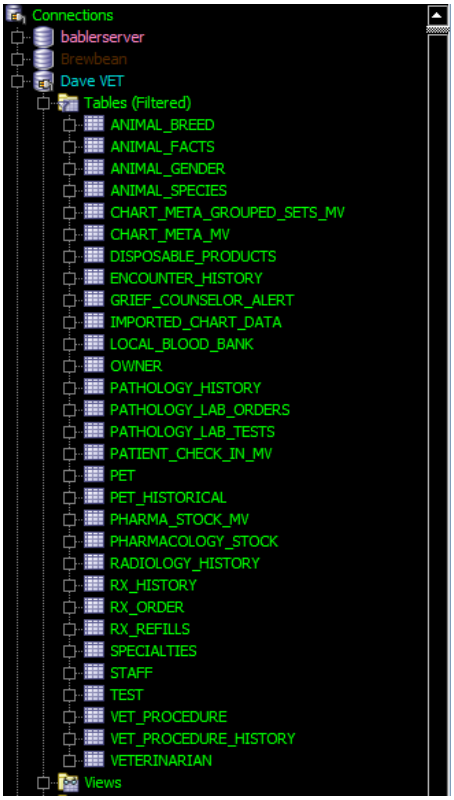


Figure 78

	TABLE_NAME	TABLESPACE_NAME	CLUSTER_NAME	IOT_NAME	STATUS	PCT_FREE	PCT_USED	INI_TRANS	MAX_TRANS	INITIAL_EXTENT	NEXT_EXTENT	MIN_EXTENTS	MAX_EXTENTS	PCT_INCREASE	NUM_ROWS	BLOCKS	EMPTY_BLOCKS	AVG_SPACE
1	STAFF	PERSONNEL	(null)	(null)	VALID	10	(null)	1	255	65536	1048576	1	2147483645	(null)	6	5	0	0
2	VETERINARIAN	PERSONNEL	(null)	(null)	VALID	10	(null)	1	255	65536	1048576	1	2147483645	(null)	4	5	0	0
3	SPECIALTIES	USERS	(null)	(null)	VALID	10	(null)	1	255	65536	1048576	1	2147483645	(null)	4	5	0	0
4	VET_PROCEDURE	PERSONNEL	(null)	(null)	VALID	10	(null)	1	255	65536	1048576	1	2147483645	(null)	11	5	0	0
5	OWNER	CRM	(null)	(null)	VALID	10	(null)	1	255	65536	1048576	1	2147483645	(null)	500	13	0	0
6	PET	CRM	(null)	(null)	VALID	10	(null)	1	255	65536	1048576	1	2147483645	(null)	16	5	0	0
7	PET_HISTORICAL	CRM	(null)	(null)	VALID	10	(null)	1	255	65536	1048576	1	2147483645	(null)	2	5	0	0
8	ANIMAL_GENDER	CRM	(null)	(null)	VALID	10	(null)	1	255	65536	1048576	1	2147483645	(null)	6	5	0	0
9	ANIMAL_SPECIES	CRM	(null)	(null)	VALID	10	(null)	1	255	65536	1048576	1	2147483645	(null)	4	5	0	0
10	ANIMAL_BREED	CRM	(null)	(null)	VALID	10	(null)	1	255	65536	1048576	1	2147483645	(null)	19	5	0	0
11	GRIEF_COUNSELOR_ALERT	CRM	(null)	(null)	VALID	10	(null)	1	255	65536	1048576	1	2147483645	(null)	2	5	0	0
12	ANIMAL_FACTS	CHART	(null)	(null)	VALID	10	(null)	1	255	65536	1048576	1	2147483645	(null)	16	5	0	0
13	ENCOUNTER_HISTORY	CHART	(null)	(null)	VALID	10	(null)	1	255	65536	1048576	1	2147483645	(null)	6	5	0	0
14	IMPORTED_CHART_DATA	CHART	(null)	(null)	VALID	10	(null)	1	255	65536	1048576	1	2147483645	(null)	1	5	0	0
15	PATHOLOGY_HISTORY	CHART	(null)	(null)	VALID	10	(null)	1	255	65536	1048576	1	2147483645	(null)	4	5	0	0
16	VET_PROCEDURE_HISTORY	CHART	(null)	(null)	VALID	10	(null)	1	255	65536	1048576	1	2147483645	(null)	6	5	0	0
17	RADIOLOGY_HISTORY	CHART	(null)	(null)	VALID	10	(null)	1	255	65536	1048576	1	2147483645	(null)	4	5	0	0
18	DISPOSABLE_PRODUCTS	CHEM	(null)	(null)	VALID	10	(null)	1	255	(null)	(null)	(null)	(null)	(null)	0	0	0	0
19	LOCAL_BLOOD_BANK	CHEM	(null)	(null)	VALID	10	(null)	1	255	65536	1048576	1	2147483645	(null)	82	5	0	0
20	PATHOLOGY_LAB_TESTS	CHEM	(null)	(null)	VALID	10	(null)	1	255	65536	1048576	1	2147483645	(null)	12	5	0	0
21	PHARMACOLOGY_STOCK	CHEM	(null)	(null)	VALID	10	(null)	1	255	65536	1048576	1	2147483645	(null)	31	5	0	0
22	RX_ORDER	CHEM	(null)	(null)	VALID	10	(null)	1	255	65536	1048576	1	2147483645	(null)	32	5	0	0
23	PATHOLOGY_LAB_ORDERS	CHEM	(null)	(null)	VALID	10	(null)	1	255	65536	1048576	1	2147483645	(null)	4	5	0	0
24	RX_HISTORY	CHART	(null)	(null)	VALID	10	(null)	1	255	65536	1048576	1	2147483645	(null)	32	5	0	0
25	RX_REFILLS	CHEM	(null)	(null)	VALID	10	(null)	1	255	65536	1048576	1	2147483645	(null)	7	5	0	0
26	TEST	USERS	(null)	(null)	VALID	10	(null)	1	255	65536	1048576	1	2147483645	(null)	1	5	0	0
27	CHART_META_GROUPED_SETS_MV	USERS	(null)	(null)	VALID	10	(null)	1	255	65536	1048576	1	2147483645	(null)	193	8	0	0
28	PATIENT_CHECK_IN_MV	CRM	(null)	(null)	VALID	10	(null)	1	255	65536	1048576	1	2147483645	(null)	14	4	0	0
29	PHARMA_STOCK_MV	USERS	(null)	(null)	VALID	10	(null)	1	255	65536	1048576	1	2147483645	(null)	31	4	0	0
30	CHART_META_MV	USERS	(null)	(null)	VALID	10	(null)	1	255	65536	1048576	1	2147483645	(null)	47	0	0	0

Figure 79

MATERIALIZED VIEWS

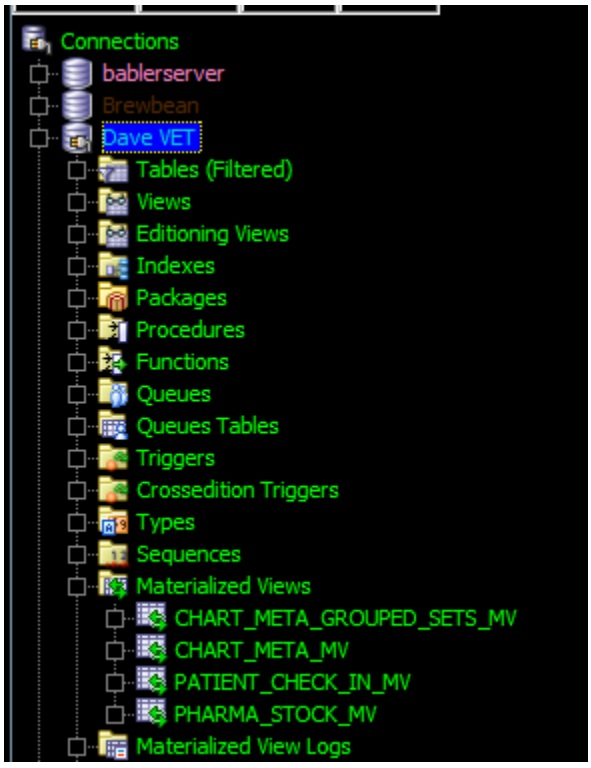


Figure 80

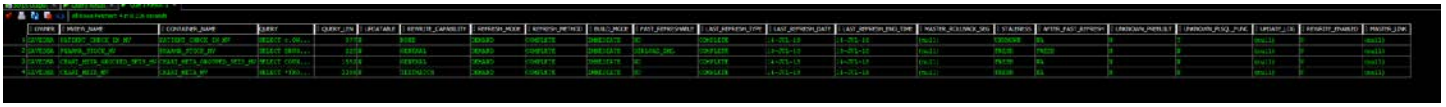


Figure 81

VIEWS

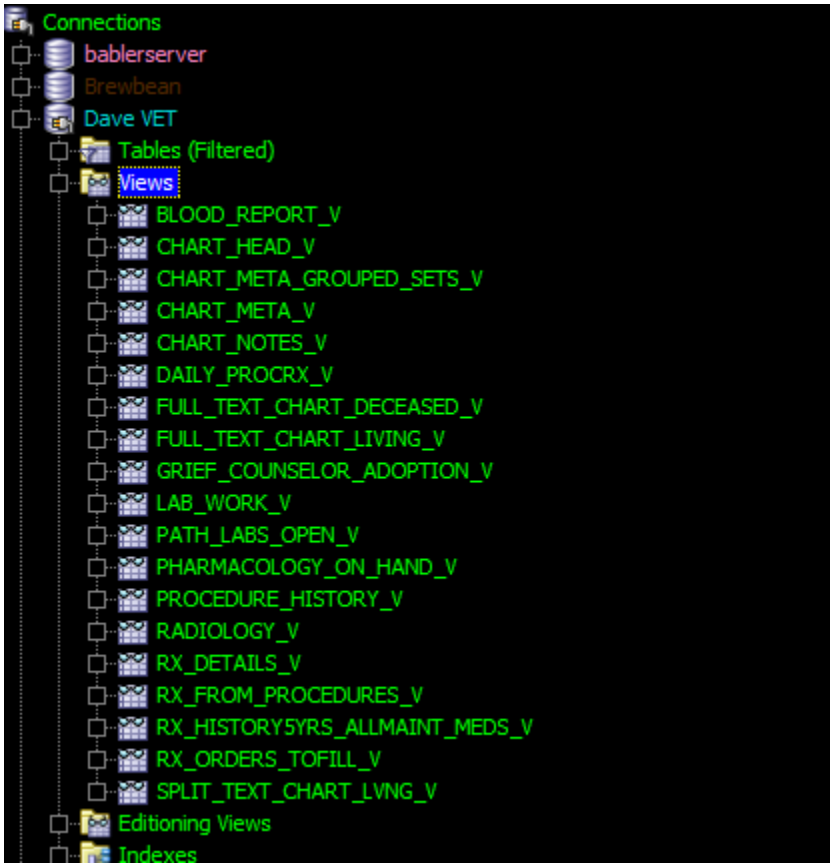


Figure 82

Script Output | Query Result

All Rows Fetched: 19 in 0.143 seconds

	OWNER	VIEW_NAME	TEXT_LENGTH	EDITIONING_VIEW	READ_ONLY	CONTAINER_DATA	BEQUEATH	TEXT	TEXT_VC
1	DAVEDBA	SPLIT_TEXT_CHART_LVNG_V	243	N	N	N	DEFINER	0	SELECT PETID, CHART_PKG.FUNC_CHART_NAME(PETID) AS PET_NAME, CHART_PKG.FUNC_CHARTHEAD(PETID) AS CHART_HEAD, CHART_
2	DAVEDBA	BLOOD_REPORT_V	145	N	N	N	DEFINER	0	SELECT COUNT(BLOODBAGID) AS TOTAL_BAGS, NVL(FUNC_SPECIES(SPECIESID), 'Grand Total') AS SPECIES FROM LOCAL_BLOOD_B
3	DAVEDBA	PHARMACOLOGY_ON_HAND_V	128	N	N	N	DEFINER	0	SELECT DRUGID, DRUG_NAME, DRUG_DOSAGE, DRUG_UNITS_INV, IS_CONTROLLED, DATE_STOCKED, DATE_EXPIRATION FROM PHAR
4	DAVEDBA	PROCEDURE_HISTORY_V	274	N	N	N	DEFINER	0	SELECT VET_PROCEDUREID, FUNC_PROCNAME(VET_PROCEDUREID) AS CLINICAL_PROCEDURE, PETID, CHART_PKG.FUNC_CHART_NAM
5	DAVEDBA	RX_HISTORY5YRS_ALLMAINT_MEDS_V	218	N	N	N	DEFINER	0	SELECT PETID, FUNC_DRUGNAME(DRUGID) as DRUG_NAME, DRUG_DOSAGE, DATE_WRITTEN, IS_MAINTENANCE_MEDFROM RX_HISTORY W
6	DAVEDBA	CHART_META_GROUPED_SETS_V	755	N	N	N	DEFINER	0	SELECT PETID, CHART_PKG.FUNC_CHART_NAME(PETID) AS PATIENT_NAME, FUNC_OWNER_NAME(PETID) AS OWNER_NAME, FUNC_ALL_SI
7	DAVEDBA	LAB_WORK_V	162	N	N	N	DEFINER	0	SELECT PETID, CHART_PKG.FUNC_CHART_NAME(PETID) AS PET_NAME, FUNC_LAB_NAME(LABID) as LAB_NAME, RESULTS, DATE_CO
8	DAVEDBA	CHART_META_V	742	N	N	N	DEFINER	0	SELECT PETID, CHART_PKG.FUNC_CHART_NAME(PETID) AS PATIENT_NAME, FUNC_OWNER_NAME(PETID) AS OWNER_NAME, FUNC_ALL_SI
9	DAVEDBA	FULL_TEXT_CHART_DECEASED_V	117	N	N	N	DEFINER	0	SELECT PETID, CHART_PKG.FUNC_FLTXTCHART(PETID) AS CHARTFROM ANIMAL_FACTSWHERE DEATH_DATE IS NOT NULLORDER BY PETI
10	DAVEDBA	FULL_TEXT_CHART_LIVING_V	113	N	N	N	DEFINER	0	SELECT PETID, CHART_PKG.FUNC_FLTXTCHART(PETID) AS CHARTFROM ANIMAL_FACTSWHERE DEATH_DATE IS NULLORDER BY PETID
11	DAVEDBA	RX_DETAILS_V	736	N	N	N	DEFINER	0	SELECT rh.RXID, rh.PETID, CHART_PKG.FUNC_CHART_NAME(rh.PETID) AS PET_NAME, FUNC_VET_NAME(rh.VETID) AS VET_NAME,
12	DAVEDBA	PATH_LABS_OPEN_V	294	N	N	N	DEFINER	0	SELECT LABORDERID, LABID, FUNC_LAB_NAME(LABID) AS "Lab Name", FUNC_CHECK_IN_NAME(PETID) AS "For", PETID AS "Patie
13	DAVEDBA	DAILY_PROCRX_V	493	N	N	N	DEFINER	0	SELECT vph.PETID, FUNC_CHECK_IN_NAME(vph.PETID) AS "Pet Name", vph.VETID, FUNC_VET_NAME(vph.VETID) AS "Veterinar
14	DAVEDBA	GRIEF_COUNSELOR_ADOPTION_V	582	N	N	N	DEFINER	0	SELECT ALERTID, ALERT_DATE, gca.PETID, FUNC_CHECK_IN_NAME(gca.PETID) as "Deceased Name", DEATH_DATE AS "Date of D
15	DAVEDBA	RADIOLOGY_V	150	N	N	N	DEFINER	0	SELECT PETID, CHART_PKG.FUNC_CHART_NAME(PETID) AS PET_NAME,RADIMG_DATE_TAKEN AS IMG_DATE, RADIMG_NOTES, RADIMG_FI
16	DAVEDBA	CHART_NOTES_V	877	N	N	N	DEFINER	0	SELECT PETID AS PETID, TO_CLOB(FUNC_VET_NAME(VETID)) AS VET, TRUNC(DATE_COMPLETED) as "EVENT_DATE", TO_CLOB(FUNC_
17	DAVEDBA	CHART_HEAD_V	319	N	N	N	DEFINER	0	SELECT PETID, CHART_PKG.FUNC_CHART_NAME(PETID) AS "Animal Name", FUNC_SPECIES(SPECIESID) AS "Species", FUNC_BREED
18	DAVEDBA	RX_FROM_PROCEDURES_V	716	N	N	N	DEFINER	0	SELECT RXID, FUNC_VET_NAME(VETID) AS "Written by", vp.VET_PROCEDURE_NAME AS "Clinical Procedure", FUNC_CHECK_IN_
19	DAVEDBA	RX_ORDERS_TOFILL_V	600	N	N	N	DEFINER	0	SELECT RXID, FUNC_VET_NAME(VETID) AS "Written by", FUNC_CHECK_IN_NAME(PETID) AS "For", FUNC_OWNER_NAME(petid) AS

Figure 83

PROGRAM UNITS

TRIGGERS

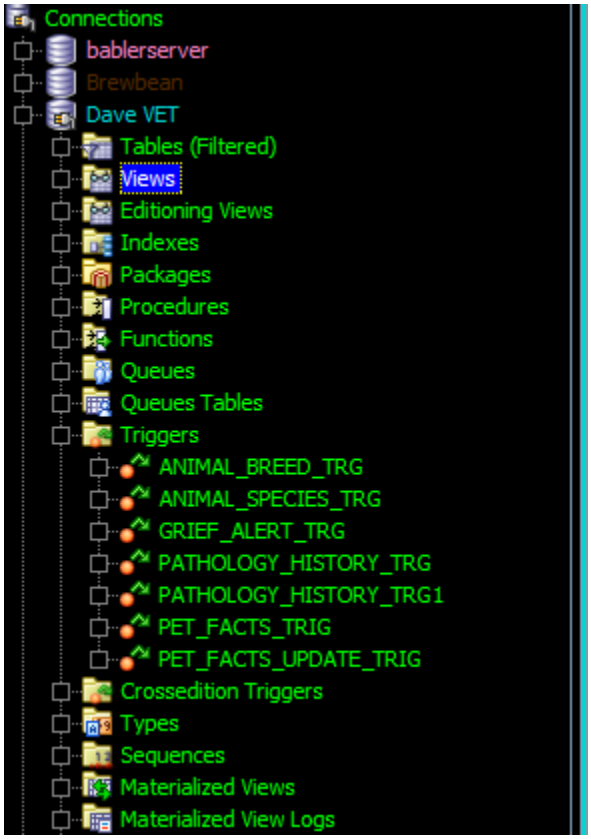


Figure 84

Script Output | Query Result

All Rows Fetched: 7 in 0.808 seconds

	OWNER	TRIGGER_NAME	TRIGGER_TYPE	TRIGGERING_EVENT	TABLE_OWNER	BASE_OBJECT_TYPE	TABLE_NAME	WHEN_CLAUSE	STATUS	DESCRIPTION
1	DAVEDBA	PET_FACTS_UPDATE_TRIG	AFTER STATEMENT	UPDATE	DAVEDBA	TABLE	PET	(null)	ENABLED	PET_FACTS_UPDATE_TRIG AFTER UPDATE OF BIRTH_DATE,BREEDID,COLORI
2	DAVEDBA	PET_FACTS_TRIG	AFTER STATEMENT	INSERT	DAVEDBA	TABLE	PET	(null)	ENABLED	pet_facts_trigAFTER INSERT ON PET
3	DAVEDBA	GRIEF_ALERT_TRG	AFTER EACH ROW	INSERT	DAVEDBA	TABLE	PET_HISTORICAL	NEW.IS_LIVING IN ('N','0', 'n')	ENABLED	GRIEF_ALERT_TRGAFTER INSERT ON PET_HISTORICALFOR EACH ROW
4	DAVEDBA	PATHOLOGY_HISTORY_TRG1	BEFORE EACH ROW	INSERT	DAVEDBA	TABLE	PATHOLOGY_HISTORY	(null)	ENABLED	PATHOLOGY_HISTORY_TRG1 BEFORE INSERT ON PATHOLOGY_HISTORY FOR E
5	DAVEDBA	PATHOLOGY_HISTORY_TRG	BEFORE EACH ROW	INSERT	DAVEDBA	TABLE	PATHOLOGY_HISTORY	(null)	ENABLED	PATHOLOGY_HISTORY_TRG BEFORE INSERT ON PATHOLOGY_HISTORY FOR EA
6	DAVEDBA	ANIMAL_SPECIES_TRG	BEFORE EACH ROW	INSERT	DAVEDBA	TABLE	ANIMAL_SPECIES	(null)	ENABLED	ANIMAL_SPECIES_TRG BEFORE INSERT ON ANIMAL_SPECIES FOR EACH ROW
7	DAVEDBA	ANIMAL_BREED_TRG	BEFORE EACH ROW	INSERT	DAVEDBA	TABLE	ANIMAL_BREED	(null)	ENABLED	ANIMAL_BREED_TRG BEFORE INSERT ON ANIMAL_BREED FOR EACH ROW

Figure 85

FUNCTIONS

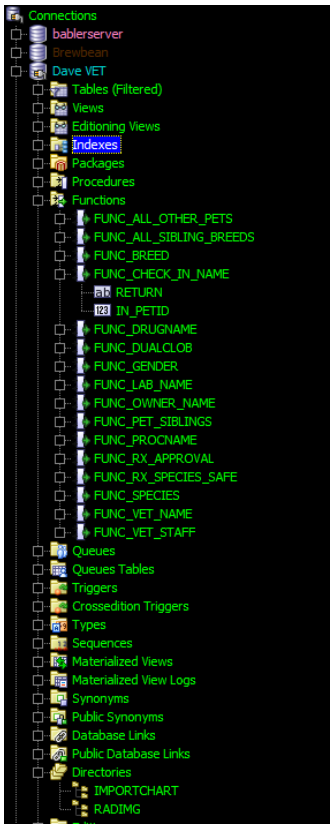


Figure 86

	OBJECT_NAME	OBJECT_ID	OBJECT_TYPE	CREATED	LAST_DDL_TIME	TIMESTAMP	STATUS	TEMPORARY	GENERATED	SECONDARY	NAMESPACE	EDITIONABLE
1	FUNC_VET_NAME	92988	FUNCTION	26-JUN-18	04-JUL-18	2018-07-04:20:55:46	VALID	N	N	N	1	Y
2	FUNC_BREED	93859	FUNCTION	06-JUL-18	08-JUL-18	2018-07-08:14:47:05	VALID	N	N	N	1	Y
3	FUNC_SPECIES	93860	FUNCTION	06-JUL-18	08-JUL-18	2018-07-08:14:47:50	VALID	N	N	N	1	Y
4	FUNC_GENDER	93861	FUNCTION	06-JUL-18	08-JUL-18	2018-07-08:14:47:28	VALID	N	N	N	1	Y
5	FUNC_PROCNAME	93865	FUNCTION	06-JUL-18	07-JUL-18	2018-07-07:15:02:20	VALID	N	N	N	1	Y
6	FUNC_DUALCLOB	93868	FUNCTION	06-JUL-18	06-JUL-18	2018-07-06:14:27:23	VALID	N	N	N	1	Y
7	FUNC_VET_STAFF	93869	FUNCTION	06-JUL-18	06-JUL-18	2018-07-06:14:55:25	VALID	N	N	N	1	Y
8	FUNC_RX_APPROVAL	93151	FUNCTION	28-JUN-18	04-JUL-18	2018-06-28:20:35:07	VALID	N	N	N	1	Y
9	FUNC_RX_SPECIES_SAFE	94450	FUNCTION	13-JUL-18	13-JUL-18	2018-07-13:15:46:39	VALID	N	N	N	1	Y
10	FUNC_ALL_OTHER_PETS	93695	FUNCTION	04-JUL-18	08-JUL-18	2018-07-08:12:23:11	VALID	N	N	N	1	Y
11	FUNC_ALL_SIBLING_BREEDS	93696	FUNCTION	04-JUL-18	08-JUL-18	2018-07-08:12:32:13	VALID	N	N	N	1	Y
12	FUNC_PET_SIBLINGS	92966	FUNCTION	25-JUN-18	10-JUL-18	2018-07-10:15:08:58	VALID	N	N	N	1	Y
13	FUNC_LAB_NAME	93531	FUNCTION	02-JUL-18	08-JUL-18	2018-07-08:14:48:09	VALID	N	N	N	1	Y
14	FUNC_CHECK_IN_NAME	92816	FUNCTION	24-JUN-18	14-JUL-18	2018-07-14:16:16:15	VALID	N	N	N	1	Y
15	FUNC_OWNER_NAME	92818	FUNCTION	24-JUN-18	08-JUL-18	2018-07-08:14:49:37	VALID	N	N	N	1	Y
16	FUNC_DRUGNAME	94101	FUNCTION	08-JUL-18	14-JUL-18	2018-07-14:16:15:01	VALID	N	N	N	1	Y

Figure 87

PROCEDURES

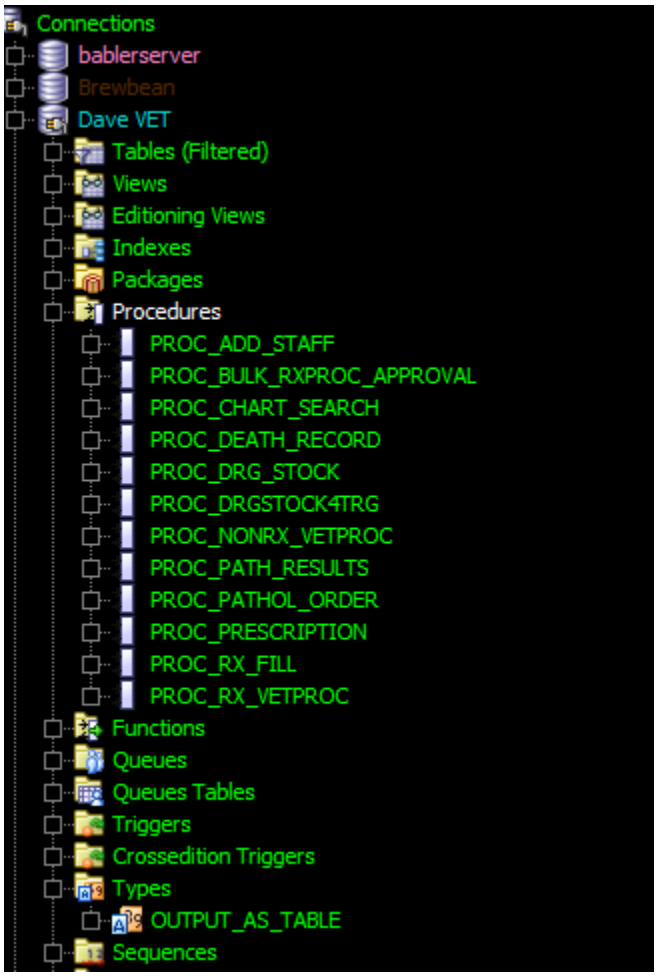


Figure 88

Script Output | Query Result

All Rows Fetched: 12 in 0.115 seconds

	OBJECT_NAME	OBJECT_ID	OBJECT_TYPE	CREATED	LAST_DDL_TIME	TIMESTAMP	STATUS	TEMPORARY	GENERATED	SECONDARY	NAMESPACE	EDITIONABLE
1	PROC_ADD_STAFF	93348	PROCEDURE	30-JUN-18	30-JUN-18	2018-06-30:15:52:06	VALID	N	N	N	1	Y
2	PROC_BULK_RXPROC_APPROVAL	93506	PROCEDURE	01-JUL-18	14-JUL-18	2018-07-14:21:20:51	VALID	N	N	N	1	Y
3	PROC_CHART_SEARCH	94794	PROCEDURE	15-JUL-18	15-JUL-18	2018-07-15:15:05:15	VALID	N	N	N	1	Y
4	PROC_DEATH_RECORD	93562	PROCEDURE	03-JUL-18	04-JUL-18	2018-07-04:21:58:03	VALID	N	N	N	1	Y
5	PROC_DRGSTOCK4TRG	93388	PROCEDURE	01-JUL-18	01-JUL-18	2018-07-01:17:18:52	VALID	N	N	N	1	Y
6	PROC_DRG_STOCK	93181	PROCEDURE	29-JUN-18	29-JUN-18	2018-06-29:14:59:15	VALID	N	N	N	1	Y
7	PROC_NONRX_VETPROC	93350	PROCEDURE	30-JUN-18	30-JUN-18	2018-06-30:21:55:23	VALID	N	N	N	1	Y
8	PROC_PATHOL_ORDER	93527	PROCEDURE	02-JUL-18	03-JUL-18	2018-07-03:11:39:28	VALID	N	N	N	1	Y
9	PROC_PATH_RESULTS	93530	PROCEDURE	02-JUL-18	03-JUL-18	2018-07-03:11:39:26	VALID	N	N	N	1	Y
10	PROC_PRESCRIPTION	93135	PROCEDURE	27-JUN-18	12-JUL-18	2018-07-12:20:38:37	VALID	N	N	N	1	Y
11	PROC_RX_FILL	93182	PROCEDURE	29-JUN-18	12-JUL-18	2018-07-12:20:37:11	VALID	N	N	N	1	Y
12	PROC_RX_VETPROC	93351	PROCEDURE	30-JUN-18	07-JUL-18	2018-07-07:15:33:04	VALID	N	N	N	1	Y

Figure 89

PACKAGES

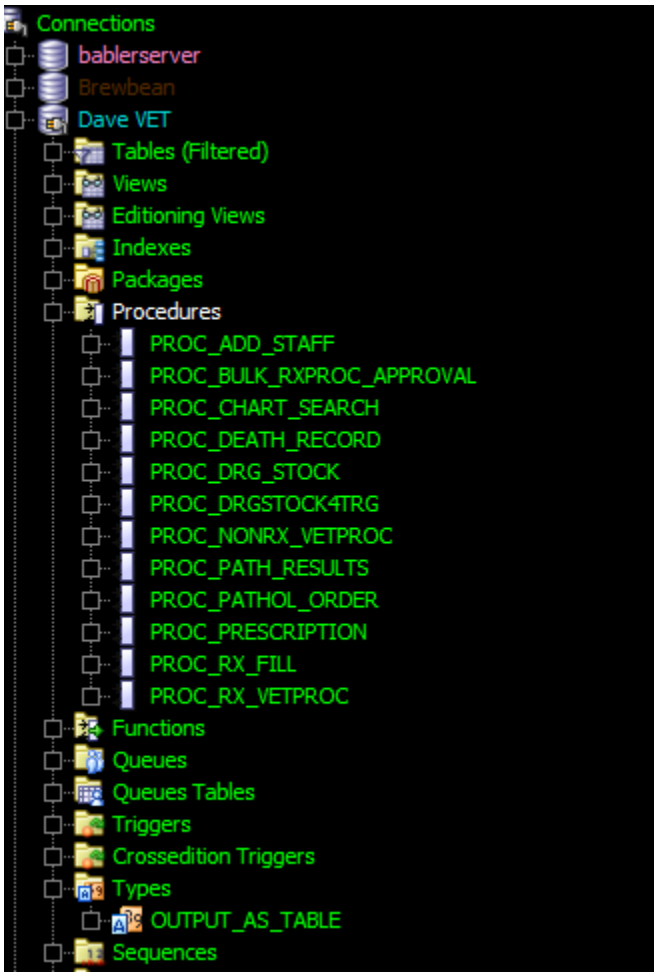


Figure 90

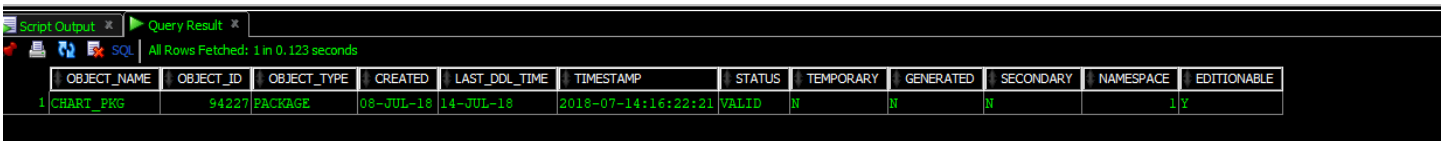


Figure 91

SEQUENCES & INDICES

SEQUENCES

Figure 92

Script Output | Query Result

SQL | All Rows Fetched: 25 in 0.117 seconds

	OBJECT_NAME	OBJECT_ID	OBJECT_TYPE	CREATED	LAST_DDL_TIME	TIMESTAMP	STATUS	TEMPORARY	GENERATED	SECONDARY	NAMESPACE	EDITIONABLE
1	ISEQ$$_92515	93347	SEQUENCE	30-JUN-18	30-JUN-18	2018-06-30:15:36:58	VALID	N	Y	N	1	(null)
2	PRESCRIPTION_SEQ	92986	SEQUENCE	26-JUN-18	26-JUN-18	2018-06-26:12:30:43	VALID	N	N	N	1	(null)
3	ISEQ$$_92618	94435	SEQUENCE	12-JUL-18	12-JUL-18	2018-07-12:20:21:45	VALID	N	Y	N	1	(null)
4	ISEQ$$_92535	92536	SEQUENCE	22-JUN-18	22-JUN-18	2018-06-22:12:35:38	VALID	N	Y	N	1	(null)
5	ISEQ$$_92540	92541	SEQUENCE	22-JUN-18	22-JUN-18	2018-06-22:12:43:02	VALID	N	Y	N	1	(null)
6	ISEQ$$_92543	92544	SEQUENCE	22-JUN-18	22-JUN-18	2018-06-22:12:46:48	VALID	N	Y	N	1	(null)
7	ISEQ$$_92561	92562	SEQUENCE	22-JUN-18	22-JUN-18	2018-06-22:12:46:48	VALID	N	Y	N	1	(null)
8	ISEQ$$_92579	92580	SEQUENCE	22-JUN-18	22-JUN-18	2018-06-22:13:17:46	VALID	N	Y	N	1	(null)
9	ISEQ$$_92584	92585	SEQUENCE	22-JUN-18	22-JUN-18	2018-06-22:13:17:46	VALID	N	Y	N	1	(null)
10	ISEQ$$_92590	92591	SEQUENCE	22-JUN-18	22-JUN-18	2018-06-22:13:17:47	VALID	N	Y	N	1	(null)
11	ISEQ$$_92597	92598	SEQUENCE	22-JUN-18	22-JUN-18	2018-06-22:13:20:29	VALID	N	Y	N	1	(null)
12	ISEQ$$_92603	92604	SEQUENCE	22-JUN-18	22-JUN-18	2018-06-22:14:14:20	VALID	N	Y	N	1	(null)
13	ISEQ$$_92606	92607	SEQUENCE	22-JUN-18	22-JUN-18	2018-06-22:14:14:20	VALID	N	Y	N	1	(null)
14	ISEQ$$_92609	92610	SEQUENCE	22-JUN-18	22-JUN-18	2018-06-22:14:14:20	VALID	N	Y	N	1	(null)
15	ISEQ$$_92623	92624	SEQUENCE	22-JUN-18	22-JUN-18	2018-06-22:14:14:59	VALID	N	Y	N	1	(null)
16	ISEQ$$_92626	92627	SEQUENCE	22-JUN-18	22-JUN-18	2018-06-22:14:16:22	VALID	N	Y	N	1	(null)
17	BREEDID_SEQ	92629	SEQUENCE	22-JUN-18	22-JUN-18	2018-06-22:14:34:03	VALID	N	N	N	1	(null)
18	SPECIESID_SEQ	92630	SEQUENCE	22-JUN-18	22-JUN-18	2018-06-22:14:34:03	VALID	N	N	N	1	(null)
19	GENDERID_SEQ	92631	SEQUENCE	22-JUN-18	22-JUN-18	2018-06-22:14:34:03	VALID	N	N	N	1	(null)
20	ANIMAL_SPECIES_SEQ	92632	SEQUENCE	22-JUN-18	22-JUN-18	2018-06-22:14:37:53	VALID	N	N	N	1	(null)
21	ANIMAL_BREED_SEQ	92633	SEQUENCE	22-JUN-18	22-JUN-18	2018-06-22:14:42:42	VALID	N	N	N	1	(null)
22	LABORDERID_SEQ	93525	SEQUENCE	02-JUL-18	02-JUL-18	2018-07-02:20:48:44	VALID	N	N	N	1	(null)
23	PATHOLOGY_HISTORY_SEQ	93550	SEQUENCE	02-JUL-18	02-JUL-18	2018-07-02:22:15:28	VALID	N	N	N	1	(null)
24	ISEQ$$_92546	92817	SEQUENCE	24-JUN-18	24-JUN-18	2018-06-24:12:09:44	VALID	N	Y	N	1	(null)
25	ISEQ$$_92615	92819	SEQUENCE	24-JUN-18	24-JUN-18	2018-06-24:13:23:57	VALID	N	Y	N	1	(null)

Figure 93

INDICES

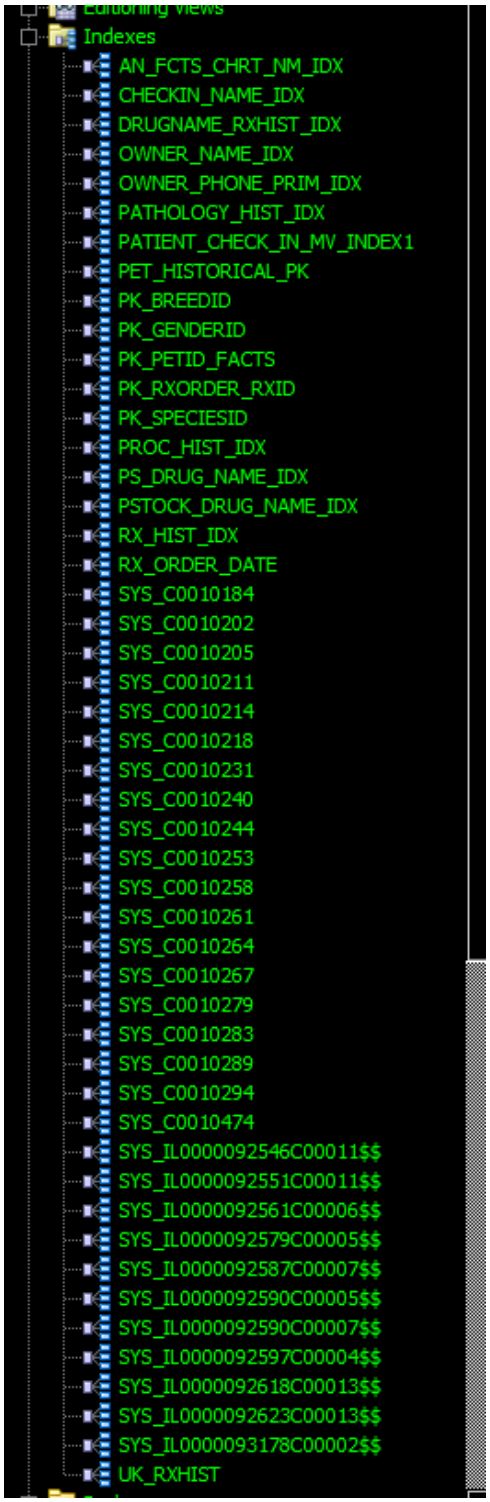


Figure 94

OBJECT_NAME	OBJECT_ID	OBJECT_TYPE	CREATED	LAST_DDL_TIME	TIMESTAMP	STATUS	TEMPORARY	GENERA...	SECONDARY	NAMESPACE	EDITIONABLE
PK_GENDERID	92556	INDEX	22-JUN-18	22-JUN-18	2018-06-22:12:46:4	VALID	N	N	N	4	(null)
UK_RXHIST	93919	INDEX	07-JUL-18	07-JUL-18	2018-07-07:13:19:0	VALID	N	N	N	4	(null)
PK_RXORDER_RXID	92619	INDEX	22-JUN-18	22-JUN-18	2018-06-22:14:14:2	VALID	N	N	N	4	(null)
PK_PETID_FACTS	92578	INDEX	22-JUN-18	22-JUN-18	2018-06-22:13:17:4	VALID	N	N	N	4	(null)
PK_BREEDID	92560	INDEX	22-JUN-18	22-JUN-18	2018-06-22:12:46:4	VALID	N	N	N	4	(null)
PK_SPECIESID	92558	INDEX	22-JUN-18	22-JUN-18	2018-06-22:12:46:4	VALID	N	N	N	4	(null)
PATIENT_CHECK_IN_MV_INDEX	94272	INDEX	10-JUL-18	10-JUL-18	2018-07-10:15:45:4	VALID	N	N	N	4	(null)
PET_HISTORICAL_PK	92554	INDEX	22-JUN-18	22-JUN-18	2018-06-22:12:46:4	VALID	N	N	N	4	(null)
RX_HIST_IDX	94634	INDEX	14-JUL-18	14-JUL-18	2018-07-14:16:36:2	VALID	N	N	N	4	(null)
PATHOLOGY_HIST_IDX	94632	INDEX	14-JUL-18	14-JUL-18	2018-07-14:16:35:0	VALID	N	N	N	4	(null)
PROC_HIST_IDX	94631	INDEX	14-JUL-18	14-JUL-18	2018-07-14:16:34:2	VALID	N	N	N	4	(null)
CHECKIN_NAME_IDX	94630	INDEX	14-JUL-18	14-JUL-18	2018-07-14:16:32:4	VALID	N	N	N	4	(null)
OWNER_NAME_IDX	94628	INDEX	14-JUL-18	14-JUL-18	2018-07-14:16:30:5	VALID	N	N	N	4	(null)
AN_FCTS_CHRT_NM_IDX	94627	INDEX	14-JUL-18	14-JUL-18	2018-07-14:16:28:4	VALID	N	N	N	4	(null)
OWNER_PHONE_PRIM_IDX	94629	INDEX	14-JUL-18	14-JUL-18	2018-07-14:16:31:2	VALID	N	N	N	4	(null)
DRUGNAME_RXHIST_IDX	94625	INDEX	14-JUL-18	14-JUL-18	2018-07-14:16:15:2	VALID	N	N	N	4	(null)
RX_ORDER_DATE	94624	INDEX	14-JUL-18	14-JUL-18	2018-07-14:16:08:1	VALID	N	N	N	4	(null)
PSTOCK_DRUG_NAME_IDX	94623	INDEX	14-JUL-18	14-JUL-18	2018-07-14:16:07:1	VALID	N	N	N	4	(null)
PS_DRUG_NAME_IDX	94622	INDEX	14-JUL-18	14-JUL-18	2018-07-14:16:05:2	VALID	N	N	N	4	(null)
SYS_C0010214	92545	INDEX	22-JUN-18	22-JUN-18	2018-06-22:12:46:4	VALID	N	Y	N	4	(null)
SYS_IL0000092618C00013$$	93177	INDEX	29-JUN-18	30-JUN-18	2018-06-30:15:19:5	VALID	N	Y	N	4	(null)
SYS_IL0000092546C00011$$	92549	INDEX	22-JUN-18	22-JUN-18	2018-06-22:12:46:4	VALID	N	Y	N	4	(null)
SYS_IL0000092561C00006$$	92564	INDEX	22-JUN-18	03-JUL-18	2018-07-03:11:47:2	VALID	N	Y	N	4	(null)
SYS_IL0000092551C00011$$	92553	INDEX	22-JUN-18	22-JUN-18	2018-06-22:12:46:4	VALID	N	Y	N	4	(null)
SYS_C0010283	92622	INDEX	22-JUN-18	02-JUL-18	2018-06-22:14:14:5	VALID	N	Y	N	4	(null)
SYS_C0010289	92625	INDEX	22-JUN-18	22-JUN-18	2018-06-22:14:14:5	VALID	N	Y	N	4	(null)
SYS_C0010294	92628	INDEX	22-JUN-18	22-JUN-18	2018-06-22:14:16:2	VALID	N	Y	N	4	(null)
SYS_C0010231	92565	INDEX	22-JUN-18	22-JUN-18	2018-06-22:12:46:4	VALID	N	Y	N	4	(null)
SYS_IL0000092587C00007$$	93529	INDEX	02-JUL-18	02-JUL-18	2018-07-02:21:14:0	VALID	N	Y	N	4	(null)
SYS_C0010240	92583	INDEX	22-JUN-18	22-JUN-18	2018-06-22:13:17:4	VALID	N	Y	N	4	(null)
SYS_C0010211	92542	INDEX	22-JUN-18	22-JUN-18	2018-06-22:12:43:0	VALID	N	Y	N	4	(null)
SYS_C0010205	92537	INDEX	22-JUN-18	22-JUN-18	2018-06-22:12:35:3	VALID	N	Y	N	4	(null)
SYS_C0010202	92534	INDEX	22-JUN-18	22-JUN-18	2018-06-22:12:34:4	VALID	N	Y	N	4	(null)
SYS_C0010184	92517	INDEX	22-JUN-18	22-JUN-18	2018-06-22:12:23:0	VALID	N	Y	N	4	(null)
SYS_C0010279	92617	INDEX	22-JUN-18	22-JUN-18	2018-06-22:14:14:2	VALID	N	Y	N	4	(null)
SYS_C0010267	92611	INDEX	22-JUN-18	22-JUN-18	2018-06-22:14:14:2	VALID	N	Y	N	4	(null)
SYS_C0010264	92608	INDEX	22-JUN-18	22-JUN-18	2018-06-22:14:14:2	VALID	N	Y	N	4	(null)
SYS_C0010261	92605	INDEX	22-JUN-18	22-JUN-18	2018-06-22:14:14:2	VALID	N	Y	N	4	(null)
SYS_IL0000092597C00004$$	92600	INDEX	22-JUN-18	22-JUN-18	2018-06-22:13:20:2	VALID	N	Y	N	4	(null)
SYS_C0010258	92601	INDEX	22-JUN-18	22-JUN-18	2018-06-22:13:20:2	VALID	N	Y	N	4	(null)
SYS_IL0000092590C00005$$	92593	INDEX	22-JUN-18	22-JUN-18	2018-06-22:13:17:4	VALID	N	Y	N	4	(null)
SYS_IL0000092590C00007$$	92595	INDEX	22-JUN-18	22-JUN-18	2018-06-22:13:17:4	VALID	N	Y	N	4	(null)
SYS_C0010253	92596	INDEX	22-JUN-18	22-JUN-18	2018-06-22:13:17:4	VALID	N	Y	N	4	(null)
SYS_C0010244	92586	INDEX	22-JUN-18	22-JUN-18	2018-06-22:13:17:4	VALID	N	Y	N	4	(null)
SYS_IL0000092579C00005$$	92582	INDEX	22-JUN-18	22-JUN-18	2018-06-22:13:17:4	VALID	N	Y	N	4	(null)
SYS_IL0000093178C00002$$	93180	INDEX	29-JUN-18	29-JUN-18	2018-06-29:11:49:1	VALID	N	Y	N	4	(null)
SYS_C0010474	94477	INDEX	13-JUL-18	13-JUL-18	2018-07-13:22:44:1	VALID	N	Y	N	4	(null)

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

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_c
ost||'|'||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||'|'||SYSDA
TE||'|'||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_uni
t    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_chart
line;

    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,DATAB
ASE_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,DATAB
ASE_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,DATAB
ASE_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,DATAB
ASE_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,DATAB
ASE_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;
```