

**Course:** MSc in Computer Science (Conversion)

**Module:** COMP40725 Introduction to Relational Databases and SQL Programming

**Assignment Title:**

“Pawsitive Prices” a new Pet Store opening in Dundrum Town Centre

Database Project

**Submitted by:**

Campbell, Zachary 07382936

**Lecturer:**

Dr. Mark Scanlon

**Submission Date:** 25th April 2014

Table of Contents

[1.1 Project Description 2](#_Toc386235501)

[1.2. Identification of Business Rules 2](#_Toc386235502)

[1.3. Identification of Assumptions Made 3](#_Toc386235503)

[1.4. Identification of Entities, Relationships, Cardinality and Optionality 3](#_Toc386235504)

[1.5. Final ERD of the Pet Store Database 6](#_Toc386235505)

[A. Tables and Attributes Diagram underlining Primary Keys 7](#_Toc386235506)

[B. Changes since the original Final Project Specification Submission. 8](#_Toc386235507)

[2.1. Database Setup 8](#_Toc386235508)

[2.2. 4 INNER JOIN Queries with descriptions 9](#_Toc386235509)

[2.3. 6 OUTER JOIN (2 x left, 2 x full, 2 x right) Queries with Descriptions 13](#_Toc386235510)

[2.4. 1 CUBE Query – Description 19](#_Toc386235511)

[2.5. 5 Sub-Queries 20](#_Toc386235512)

[2.6. 5 PL/SQL procedures as part of one package. 25](#_Toc386235513)

[2.7. 2 PL/SQL Functions 32](#_Toc386235514)

[2.8. 3 Triggers (at least 1 before, and at least 1 after) 35](#_Toc386235515)

[2.9. Identification of weaknesses or potential improvements of the database 39](#_Toc386235516)

## 1.1 Project Description

A new American pet store franchise is opening in Dundrum Town Centre later this year called “Pawsitive Prices”. The store requires a database to be created to keep track of its business. The store hopes that the data gained will allow them to implement smart data-driven business strategies in the next financial quarter. Within this database, it needs to keep track of its customer details. The store is also concerned with recording which employees work in what department and who their manager is. The database must be able to specify what each employee’s salary is and what his or her job role in the company is. The store is also only able to staff a certain amount of employees so they will have to cover different departments at different times during busy periods. Each employee will help produce customer orders. Each order in the database must contain comprehensive details about the product and supplies. All products and supplies are from a designated wholesaler in the U.K. The order details must show which animals can be ordered from the store and from which breeders they are ordered. The store also wishes to offer potential discounts on orders.

## 1.2. Identification of Business Rules

* Employees have to work in many departments.
* Employees must have a job title and job description
* Every employee must have a manager.
* Employees are responsible for processing many orders.
* It is mandatory for orders to be processed by an employee.
* It is mandatory for orders to have a customer
* Customers can issue an order in the store.
* Customers have the option of using a store discount.
* Order details must involve a product, supply, or pet.
* One order can have multiple order details
* Many pets must come from many animal breeders.
* All pet products and supplies come from one wholesaler.
* Only one discount can be used on one order.

## 1.3. Identification of Assumptions Made

* It is optional which employees work in what department.
* It is assumed that an employee must have a manager unless they are one.
* It is optional which employee processes which orders.
* It is assumed that customers can have many orders.
* It is assumed that many discounts can be applied to many different orders.
* Multiple different products, supplies, and pets can be involved in one order detail.
* It is optional for certain breeders to be involved with selling certain pets to the store.
* It is assumed that customers can be registered to the database without an order.
* It is assumed animals ordered in can have the same date of birth as they could be from the same mother.

## 1.4. Identification of Entities, Relationships, Cardinality and Optionality

During the iterative process of designing the database, many entities were excluded from the final design. The following entities were removed: Employee Salary, Loyalty Rewards, Inventory, and Wholesaler. The final entities are visible in Table 4.1.

**Table 4.1 Entities**

|  |  |
| --- | --- |
| Breeders | Order Details |
| Customers | Orders |
| Discounts | Pets |
| Employee | Products |
| Managers | Supplies |
| Inventory | Employee Role |

**Table 4.2 Relationships**

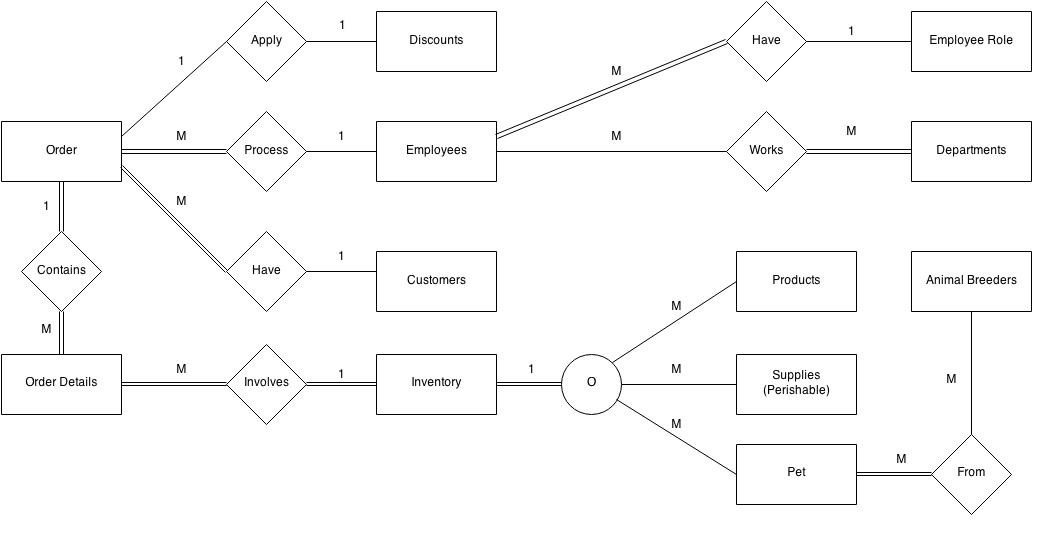
The previous relationship tables are now outdated. Please see the ERD diagram to visually see the relationships between entities. The cardinality and optionality below also highlight the relationships that entities have with each other.

**Cardinality and Optionality**

1. Many employees work in many departments and many departments have many employees. It is optional for employees to work in departments but mandatory for departments to have employees.
2. Many employees have one employee role and one employee role is given to many employees. It is mandatory for an employee to have a role but an employee role can exist without being applied to an employ.
3. Many orders are processed by one employee and one employee processes many orders. It is mandatory for orders to be processed by an employee but optional for an employee to process orders.
4. Many orders can have one customer and one customer can have many orders. It is mandatory for orders to have a customer but it is optional for a customer to have an order.
5. One order can have one discount and one discount can be applied to one order. It is optional for an order to have a discount and optional for a discount to be applied to an order.
6. Order details contain one order and one order can contain many order details. It is mandatory for order details to contain an order and for orders to contain order details.
7. Many order details involve one item from inventory. One item from inventory can be involved in many order details.
8. One inventory item can involve many products and many products are assigned as one inventory item. It is mandatory for inventory to involve products but optional for products to be involved in inventory
9. One inventory item can involve many supplies and many supplies are assigned as one inventory item. It is mandatory for inventory to involve supplies but optional for supplies to be involved in inventory
10. One inventory item can involve many pets and many pets are assigned as one inventory item. It is mandatory for inventory to involve pets but optional for pets to be involved in inventory
11. Many pets come from many animal breeders and many animals breeders sell pets. It is mandatory for pets to be involved with breeders and it is optional for breeders to be involved with pets.

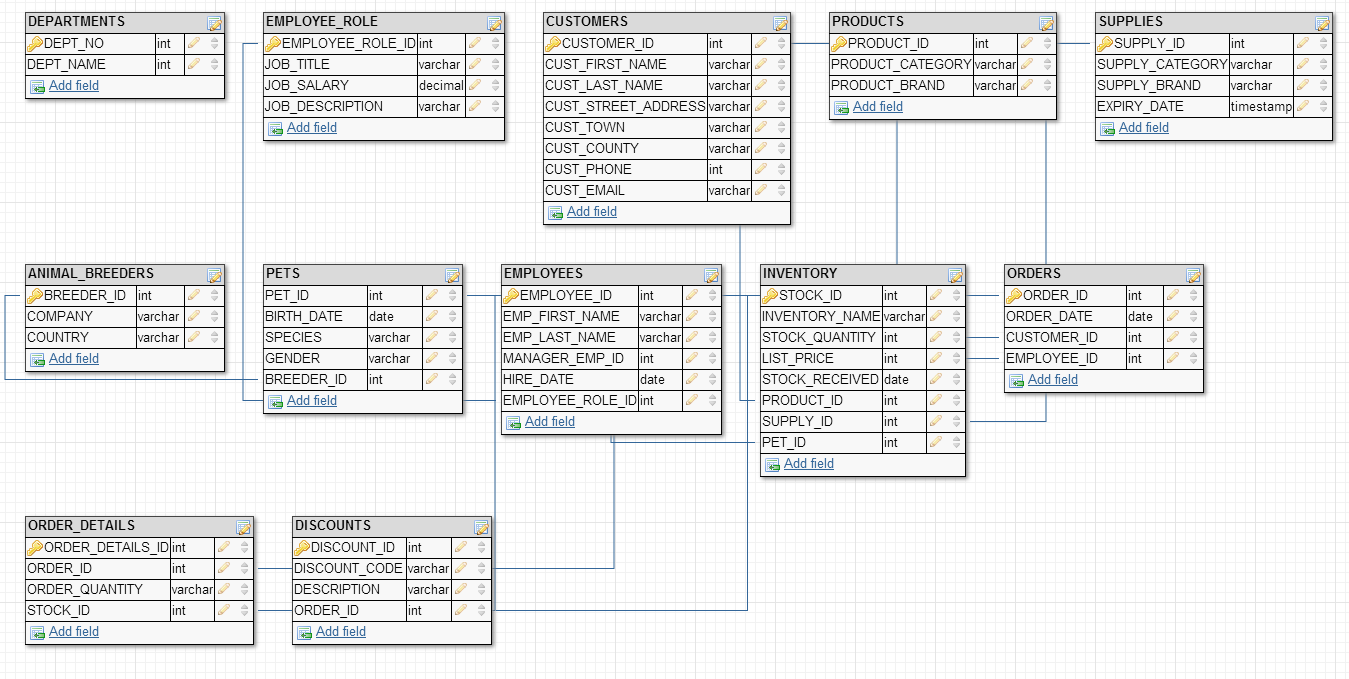
## 1,5. Final ERD of the Pet Store Database

The pet store database has received some improvements from its previous iterations. Most importantly, the issue between order details and products, supplies and pets has been resolved. An inventory table was created to rectify this. Specialisation is also now used to highlight that stock inventory can involve products, supplies, or pets. Another change was the addition of the employee role table. The table was created in order to improve database functionality by enabling more unique queries to be done.



## A. Tables and Attributes Diagram underlining Primary Keys

**PETSTORE DATABASE SCHEMA**



**Two junction tables were also created in this database:**

EMPLOYEE\_DEPARTMENTS (*EMPLOYEE\_*ID\_FK, DEPT\_NO\_FK)

PETS\_ANIMAL\_BREEDERS (PET\_ID, BREEDER\_ID)

## B. Changes since the original Final Project Specification Submission.

## 2.1. Database Setup

* Business rules have been refined as some business rules were assumptions and vice versa.
* An inventory table has been added to the database. This was to solve the issue with order details, and how products, supplies, and pets are managed.
* Visual relationships of the database were removed but the cardinality and optionality have been re-written to reflect the updated database.
* The employee role table has been added to enhance query functionality.
* The cardinality and optionality of discounts is now explained, as it was missing previously.
* The final ERD is now shown in the first part of the assignment instead of its previous iteration.

## 2.2. 4 INNER JOIN Queries with descriptions

**Inner Join 1 - Description**

List of each department each employee works in. The junction table is used to make the inner join work between the two tables and to solve the many to many problem e.g. many employees work in many departments

**Inner Query Code**

**SELECT**

**E.EMPLOYEE\_ID,**

**E.EMP\_FIRST\_NAME|| ' ' || E.EMP\_LAST\_NAME AS EMPLOYEE\_FULL\_NAME,**

**ED.DEPT\_NO**

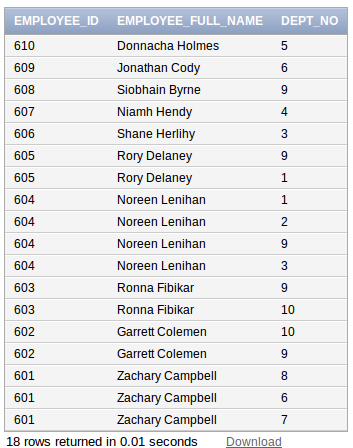
**FROM EMPLOYEES E**

**INNER JOIN EMPLOYEE\_DEPARTMENTS ED**

**ON E.EMPLOYEE\_ID = ED.EMPLOYEE\_ID**

**ORDER BY E.EMPLOYEE\_ID DESC;**

**Query Output**



**Inner Join 2 - Description**

An inner join between order details table and inventory table. This query shows the total order price of each order item by calculating the list price vs order quantity. This inner join gives a list of products, supply and pet highlighting the most expensive orders per category in a descending order.

**Inner Query Code**

**SELECT**

**OD.ORDER\_ID,**

**OD.STOCK\_ID,**

**OD.ORDER\_QUANTITY,**

**I.INVENTORY\_NAME,**

**I.LIST\_PRICE,**

**I.LIST\_PRICE\*OD.ORDER\_QUANTITY AS TOTAL\_PRICE**

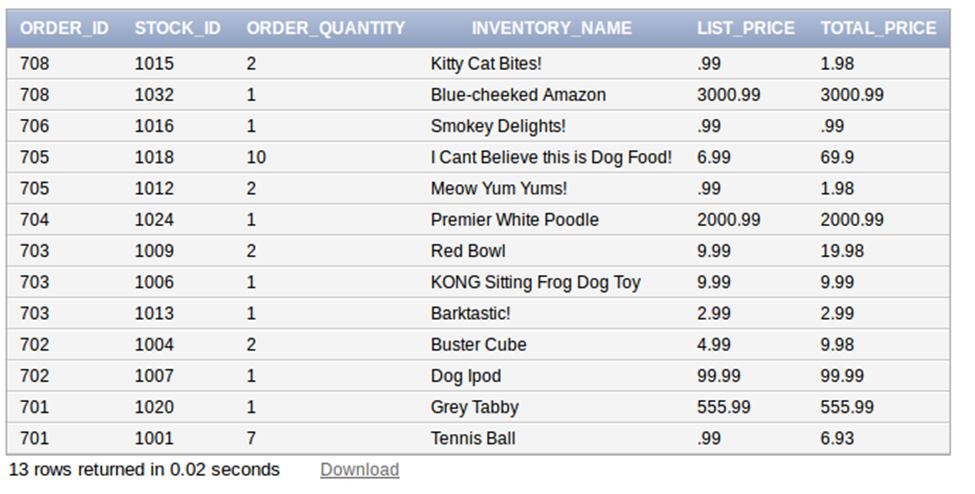
**FROM ORDER\_DETAILS OD**

**INNER JOIN INVENTORY I**

**ON OD.STOCK\_ID = I.STOCK\_ID**

**ORDER BY OD.ORDER\_ID DESC;**

**Query Output**

****

**Inner Join 3 - Description**

An inner join between the employee table and employee role. This combines essential information together to know which employee started at what time and what their title/salary is.

**Inner Query Code**

**SELECT**

**E.EMP\_FIRST\_NAME|| ' ' || E.EMP\_LAST\_NAME AS EMPLOYEE\_FULL\_NAM,**

**E.HIRE\_DATE,**

**ER.JOB\_TITLE,**

**ER.JOB\_SALARY**

**FROM EMPLOYEES E**

**INNER JOIN EMPLOYEE\_ROLE ER**

**ON E.EMPLOYEE\_ROLE\_ID = ER.EMPLOYEE\_ROLE\_ID**

**ORDER BY ER.JOB\_SALARY DESC;**

**Query Output**



**Inner Join 4 - Description**

An inner join between the pets table and animal breeders table. This allows the pet store to know the origin of each animal with the breeder identification beside it.

**Inner Query Code**

**SELECT**

**P.PET\_ID,**

**P.BIRTH\_DATE,**

**P.SPECIES,**

**P.GENDER,**

**PAB.BREEDER\_ID**

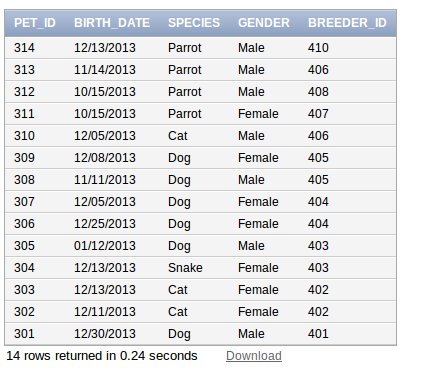
**FROM PETS P**

**INNER JOIN PETS\_ANIMAL\_BREEDERS PAB**

**ON P.PET\_ID = PAB.PET\_ID**

**ORDER BY P.PET\_ID DESC;**

**Query Output**



## 2.3. 6 OUTER JOIN (2 x left, 2 x full, 2 x right) Queries with Descriptions

**Left Outer Join 1 – Description**

This query returns employees full name with any orders they have processed and shows which employees have not yet processed orders. The list is ordered by employee first name.

**Left Outer Join Query Code**

**SELECT**

**E.EMP\_FIRST\_NAME|| ' ' || E.EMP\_LAST\_NAME AS EMPLOYEE\_FULL\_NAM,**

**O.ORDER\_DATE,**

**O.ORDER\_ID**

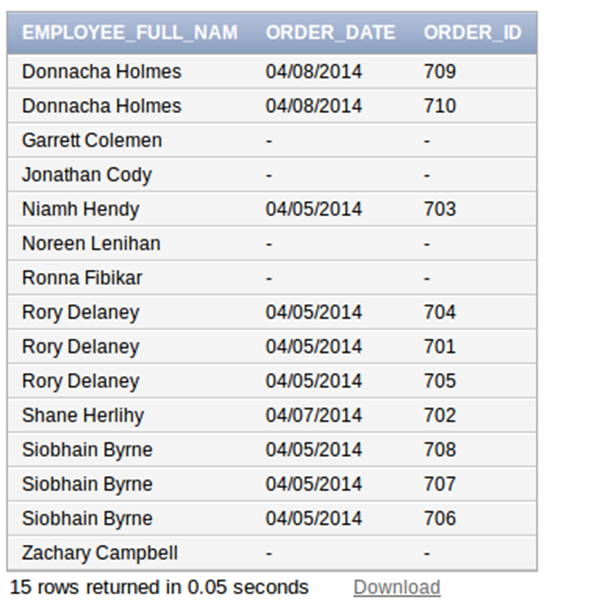
**FROM EMPLOYEES E**

**LEFT OUTER JOIN ORDERS O**

**ON E.EMPLOYEE\_ID = O.EMPLOYEE\_ID**

**ORDER BY E.EMP\_FIRST\_NAME;**

**Query Output**



**Left Outer Join 2 – Description**

This query returns the product category, with the product brand, name of the product, list price and stock id. The query highlights the fact that the store only has a limited amount of products and brands available in stock.

**Left Outer Join Query Code**

**SELECT**

**P.PRODUCT\_CATEGORY,**

**P.PRODUCT\_BRAND,**

**I.INVENTORY\_NAME,**

**I.LIST\_PRICE,**

**I.STOCK\_ID**

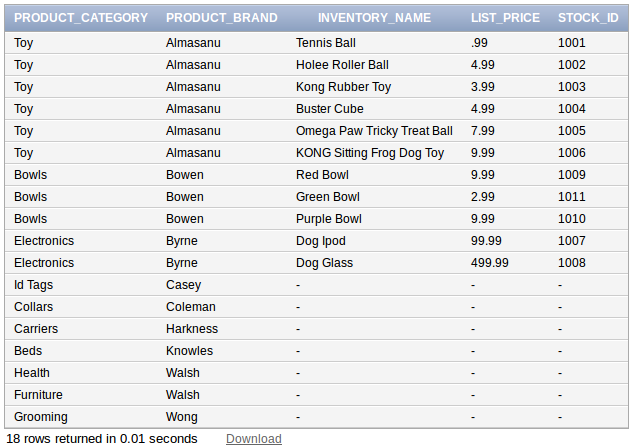
**FROM PRODUCTS P**

**LEFT OUTER JOIN INVENTORY I**

**ON P.PRODUCT\_ID = I.PRODUCT\_ID**

**ORDER BY P.PRODUCT\_BRAND ASC**

**Query Output**



**Full Outer Join 1 – Description**

This full outer join returns rows from the Customers table (left table), and rows from the Orders table (right table). This query shows all customers with or without their order number / order date. Note that customers are not required to purchase a product, supply or pet to be allowed on to the database.

**Full Outer Join Query Code**

**C.CUST\_FIRST\_NAME || ' ' || C.CUST\_LAST\_NAME AS CUSTOMER\_FULL\_NAME,**

**O.ORDER\_ID,**

**O.ORDER\_DATE**

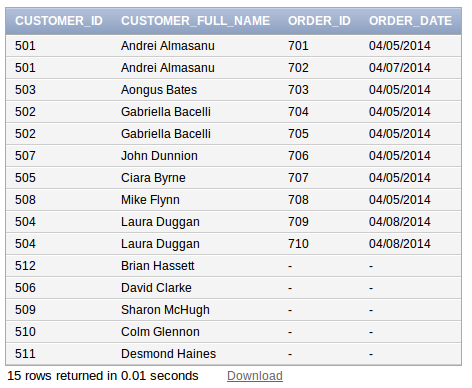
**FROM CUSTOMERS C**

**FULL OUTER JOIN ORDERS O**

**ON C.CUSTOMER\_ID = O.CUSTOMER\_ID**

**ORDER BY O.ORDER\_ID ASC;**

**Query Output**



**Full Outer Join 2 – Description**

This full outer join returns rows from the discounts table (left table) and rows from the orders table (right table). This query shows all discounts available on orders (ascending). The rows without discounts are also displayed as it is a full outer join.

**Full Outer Join Query Code**

**SELECT**

**D.DISCOUNT\_ID,**

**D.DISCOUNT\_CODE,**

**O.ORDER\_ID**

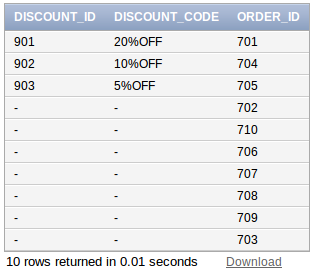
**FROM DISCOUNTS D**

**FULL OUTER JOIN ORDERS O**

**ON D.ORDER\_ID = O.ORDER\_ID**

**ORDER BY D.DISCOUNT\_ID ASC;**

**Query Output**



**Right Outer Join 1 – Description**

This right outer join displays the animals in the recorded inventory in the shop with comprehensive information.

**Right Outer Join Query Code**

**SELECT**

**I.STOCK\_ID,**

**I.INVENTORY\_NAME AS PET\_NAME,**

**P.BIRTH\_DATE,**

**P.SPECIES,**

**P.GENDER,**

**P.BREEDER\_ID**

**FROM**

**INVENTORY I**

**RIGHT OUTER JOIN PETS P**

**ON P.PET\_ID = I.PET\_ID**

**ORDER BY I.STOCK\_ID DESC;**

**Query Output**



**Right Outer Join 2 – Description**

This right outer join shows all orders placed to date alongside the customer and employee involved. The details for the order item, order quantity and stock id further enhance the information.

**Right Outer Join Query Code**

**SELECT**

**O.ORDER\_ID,**

**O.ORDER\_DATE,**

**O.CUSTOMER\_ID,**

**O.EMPLOYEE\_ID,**

**OD.ORDER\_DETAILS\_ID,**

**OD.ORDER\_ID,**

**OD.ORDER\_QUANTITY,**

**OD.STOCK\_ID**

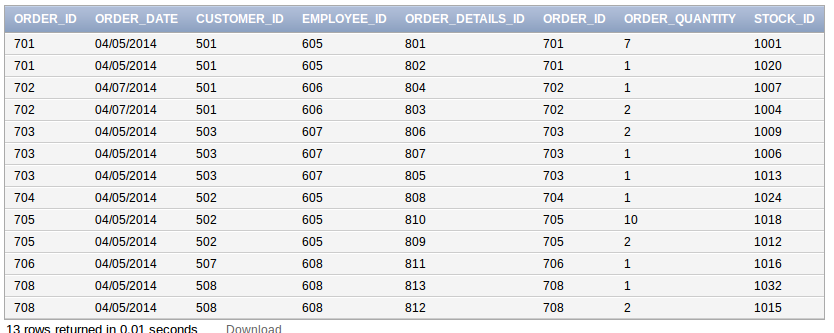
**FROM ORDERS O**

**RIGHT OUTER JOIN ORDER\_DETAILS OD**

**ON O.ORDER\_ID= OD.ORDER\_ID**

**ORDER BY O.ORDER\_ID ASC;**

**Query Output**



## 2.4. 1 CUBE Query – Description

This query displays each of the order details id, the stock id and the sum of all orders. The cube query has been grouped to show which products are selling the most with a final output of the total sales revenue

**Cube Query Code**

**SELECT**

**OD.ORDER\_DETAILS\_ID,**

**OD.STOCK\_ID,**

**SUM(I.LIST\_PRICE \* I.STOCK\_QUANTITY) AS SUM\_OF\_ALL\_ORDERS**

**FROM ORDER\_DETAILS OD**

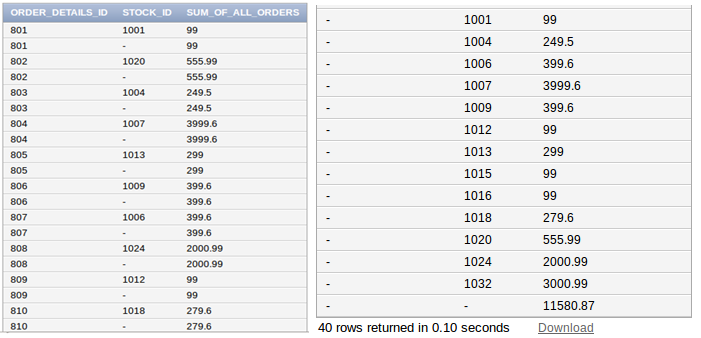
**JOIN INVENTORY I**

**ON OD.STOCK\_ID = I.STOCK\_ID**

**GROUP BY CUBE (OD.ORDER\_DETAILS\_ID,OD.STOCK\_ID)**

**ORDER BY OD.ORDER\_DETAILS\_ID, OD.STOCK\_ID;**

**Query Output**



## 2.5. 5 Sub-Queries

**Sub-Query 1 – Description**

The store wants to send out promotional leaflets to customers with addresses in Dublin. Please note that customers are anyone who have come into the store and offered to be registered by staff on to the database. This means that they may or may not have a sale.

**Query Code**

**SELECT**

**CUSTOMER\_ID,**

**CUST\_FIRST\_NAME || ' ' || CUST\_LAST\_NAME AS CUSTOMER\_FULL\_NAME**

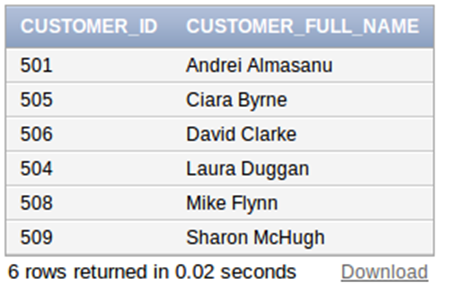
**FROM**

**CUSTOMERS**

**WHERE CUST\_COUNTY ='Dublin'**

**ORDER BY CUSTOMER\_FULL\_NAME;**

**Query Output**



**Sub-Query 2 – Description**

The store manager wants to order more male/female cats from the animal breeders but wants to balance the gender ratio out with dogs. The following query gives him a quick count.

**Query Code**

**SELECT**

**SPECIES,**

**GENDER,**

**COUNT(\*)**

**FROM PETS**

**WHERE**

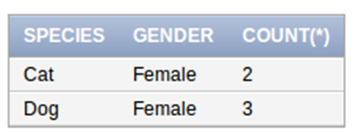
**(SPECIES = 'Cat' AND GENDER = 'Female') OR**

**(SPECIES = 'Dog' AND GENDER = 'Female')**

**GROUP BY SPECIES, GENDER**

**ORDER BY SPECIES;**

**Query Output**



**Sub-Query 3 – Description**

The store manager is suspicious of staff handing out the 20% discount code and applying it to too many orders. The query below shows that the manager had nothing to worry about after all!

**Query Code**

**SELECT**

**D.DISCOUNT\_ID,**

**D.DISCOUNT\_CODE,**

**D.DESCRIPTION,**

**O.ORDER\_ID**

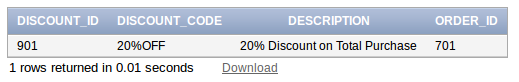
**FROM DISCOUNTS D**

**JOIN ORDERS O**

**ON D.ORDER\_ID=O.ORDER\_ID**

**WHERE D.DISCOUNT\_CODE = '20%OFF';**

**Query Output**



**Sub-Query 4 – Description**

The store manager wants to know what is the most valuable item in the stores inventory i.e if its a Product, Supply or Pet \*/

**Query Code**

**SELECT**

**STOCK\_ID,**

**INVENTORY\_NAME,**

**LIST\_PRICE,**

**PRODUCT\_ID,**

**SUPPLY\_ID,**

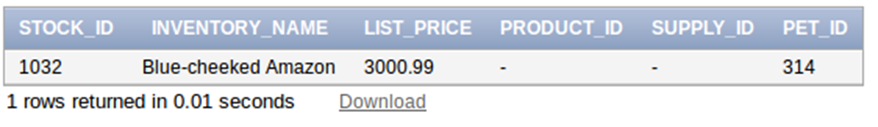
**PET\_ID**

**FROM**

**INVENTORY**

**WHERE LIST\_PRICE = (SELECT MAX(LIST\_PRICE) FROM INVENTORY);**

**Query Output**



**Sub-Query 5 – Description**

The store manager wants to find out the lowest paid staff in the company. He will then review each of the candidates individually to see who is ambitious enough for the promotion.

**Query Code**

**SELECT**

**E.EMPLOYEE\_ID,**

**E.EMP\_FIRST\_NAME,**

**E.EMP\_LAST\_NAME,**

**ER.JOB\_SALARY,**

**ER.JOB\_TITLE**

**FROM**

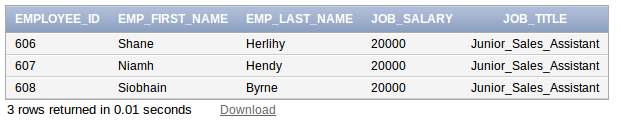
**EMPLOYEES E**

**JOIN EMPLOYEE\_ROLE ER**

**ON E.EMPLOYEE\_ROLE\_ID = ER.EMPLOYEE\_ROLE\_ID**

**WHERE ER.JOB\_SALARY = (SELECT MIN(JOB\_SALARY) FROM EMPLOYEE\_ROLE);**

**Query Output**



2.6. 5 PL/SQL procedures as part of one package.

Creating package **PETSTORE\_PACKAGE** with **5 procedures**:

* **Procedure 1** - This procedure removes the last entered discount from the discounts table. This way discounts are for a limited only!
* **Procedure 2** - This procedure allows the user to look up items and the price of those items!
* **Procedure 3** - This procedure allows the user to compare inventory!
* **Procedure 4** - This procedure allows the user to see employee details!
* **Procedure 5** - This employee salary range

**PACKAGE CODE – INDIVIDUAL PROCEDURES TESTED ON SUBSEQUENT PAGES**

**CREATE OR REPLACE PACKAGE PETSTORE\_PACKAGE AS**

**PROCEDURE REMOVELAST\_DISCOUNT;**

**PROCEDURE STOCK\_LOOKUP(INPUT\_ID IN NUMBER);**

**PROCEDURE COMPARE\_INVENTORY (INPUT1 IN NUMBER, INPUT2 IN NUMBER);**

**PROCEDURE SHOW\_EMPLOYEE\_DETAILS (INPUT1 IN NUMBER);**

**--PROCEDURE EMPLOYEE\_SALARY\_RANGE;**

**END PETSTORE\_PACKAGE;**

**/**

**CREATE OR REPLACE PACKAGE BODY PETSTORE\_PACKAGE AS**

**/\* Procedure 1 - This procedure removes the last entered discount**

**from the discounts table. This way discounts are for a limited only! \*/**

**PROCEDURE REMOVELAST\_DISCOUNT**

**IS**

**BEGIN**

**-- Savepoint created before discount removal**

**SAVEPOINT BEFORE\_DISCOUNT\_REMOVAL;**

**-- Selecting where to delete from \*/**

**DELETE FROM DISCOUNTS**

**-- When row number is less or equal to 1**

**WHERE ROWNUM <=1;**

**-- Rollback on Error**

**-- Print statement on successfully removing oldest discount offer**

**DBMS\_OUTPUT.PUT\_LINE('SUCCESSFULLY REMOVED OLDEST DISCOUNT OFFER');**

**EXCEPTION**

**-- When others output error message**

**WHEN OTHERS THEN**

**DBMS\_OUTPUT.PUT\_LINE('WARNING! YOU HAVE FAILED TO REMOVE OLDEST DISCOUNT!');**

**-- Rollback to before discount removal**

**ROLLBACK TO BEFORE\_DISCOUNT\_REMOVAL;**

**END;**

**/\* Procedure 2 - This procedure allows the user to look up items and the price of those items! \*/**

**PROCEDURE STOCK\_LOOKUP (INPUT\_ID IN NUMBER)**

**AS**

**INPUT\_1 NUMBER := INPUT\_ID;**

**INV\_NAME VARCHAR(255);**

**PRICE NUMBER;**

**NOSTOCK EXCEPTION;**

**BEGIN**

**SELECT INVENTORY\_NAME**

**INTO INV\_NAME**

**FROM INVENTORY**

**WHERE STOCK\_ID = INPUT\_1;**

**SELECT LIST\_PRICE**

**INTO PRICE**

**FROM INVENTORY**

**WHERE STOCK\_ID = INPUT\_1;**

**DBMS\_OUTPUT.PUT\_LINE('Inventory Name - ' || INV\_NAME );**

**DBMS\_OUTPUT.PUT\_LINE('Price of Inventory: ' || PRICE );**

**EXCEPTION**

**WHEN NO\_DATA\_FOUND THEN**

**DBMS\_OUTPUT.PUT\_LINE('WARNING AN ERROR HAS OCCURED! NO INFO AS PRODUCT ID DOESNT EXIST!');**

**END;**

**/\* Procedure 3 - This procedure allows the user to compare inventory! \*/**

**PROCEDURE COMPARE\_INVENTORY (INPUT1 IN NUMBER, INPUT2 IN NUMBER)**

**AS**

**INPUT\_1 NUMBER := INPUT1;**

**INPUT\_2 NUMBER := INPUT2;**

**INV\_NAME\_1 NUMBER;**

**INV\_NAME\_2 NUMBER;**

**PRICE\_COMPARISON NUMBER;**

**BEGIN**

**SELECT LIST\_PRICE**

**INTO INV\_NAME\_1**

**FROM INVENTORY**

**WHERE STOCK\_ID = INPUT\_1;**

**SELECT LIST\_PRICE**

**INTO INV\_NAME\_2**

**FROM INVENTORY**

**WHERE STOCK\_ID = INPUT\_2;**

**SELECT INV\_NAME\_2-INV\_NAME\_1**

**INTO PRICE\_COMPARISON**

**FROM DUAL;**

**DBMS\_OUTPUT.PUT\_LINE('Inventory Item 1 : ' || INV\_NAME\_1);**

**DBMS\_OUTPUT.PUT\_LINE('Inventory Item 2: ' || INV\_NAME\_2);**

**DBMS\_OUTPUT.PUT\_LINE('Price Comparison: ' || PRICE\_COMPARISON);**

**EXCEPTION**

**WHEN NO\_DATA\_FOUND THEN**

**DBMS\_OUTPUT.PUT\_LINE('WARNING ONE OR BOTH OF THESE ITEMS DOES NOT EXIST IN THE DATABASE!');**

**END;**

**/\* Procedure 4 - This procedure allows the user to see employee details! \*/**

**PROCEDURE SHOW\_EMPLOYEE\_DETAILS (INPUT1 IN NUMBER)**

**AS**

**INPUT\_1 NUMBER := INPUT1;**

**FIRST\_NAME VARCHAR(50);**

**LAST\_NAME VARCHAR(50);**

**MANAGER\_EMPLOYEE\_ID NUMBER;**

**DATE\_HIRED VARCHAR(150);**

**JOB\_TITLE VARCHAR(100);**

**NO\_EMPLOYEE EXCEPTION;**

**BEGIN**

**SELECT EMP\_FIRST\_NAME**

**INTO FIRST\_NAME**

**FROM EMPLOYEES**

**WHERE EMPLOYEE\_ID = INPUT\_1;**

**SELECT EMP\_FIRST\_NAME**

**INTO LAST\_NAME**

**FROM EMPLOYEES**

**WHERE EMPLOYEE\_ID = INPUT\_1;**

**SELECT MANAGER\_EMP\_ID**

**INTO MANAGER\_EMPLOYEE\_ID**

**FROM EMPLOYEES**

**WHERE EMPLOYEE\_ID = INPUT\_1;**

**SELECT HIRE\_DATE**

**INTO DATE\_HIRED**

**FROM EMPLOYEES**

**WHERE EMPLOYEE\_ID = INPUT\_1;**

**SELECT ER.JOB\_TITLE**

**INTO JOB\_TITLE**

**FROM EMPLOYEE\_ROLE ER**

**JOIN EMPLOYEES E**

**ON E.EMPLOYEE\_ROLE\_ID = ER.EMPLOYEE\_ROLE\_ID**

**WHERE ER.EMPLOYEE\_ROLE\_ID = INPUT\_1;**

**DBMS\_OUTPUT.PUT\_LINE('Employee Role: ' || JOB\_TITLE);**

**DBMS\_OUTPUT.PUT\_LINE('First Name : ' || FIRST\_NAME);**

**DBMS\_OUTPUT.PUT\_LINE('Last Name: ' || LAST\_NAME);**

**DBMS\_OUTPUT.PUT\_LINE('Manager Employee ID: ' || MANAGER\_EMPLOYEE\_ID);**

**DBMS\_OUTPUT.PUT\_LINE('Date Hired: ' || DATE\_HIRED);**

**DBMS\_OUTPUT.PUT\_LINE('Job Title: ' || JOB\_TITLE);**

**EXCEPTION**

**WHEN NO\_DATA\_FOUND THEN**

**DBMS\_OUTPUT.PUT\_LINE('WARNING THIS EMPLOYEE DOES NOT EXIST IN DATABASE!');**

**END;**

**/\***

**PROCEDURE EMPLOYEE\_SALARY\_RANGE**

**AS**

**DECLARE**

**-- Cursor accessing all employees by id and salary from employee roles table**

**CURSOR EMPLOYEE\_CURSOR IS SELECT EMPLOYEE\_ROLE\_ID, JOB\_SALARY FROM EMPLOYEE\_ROLE;**

**-- Creating the row type cursor**

**EMPLOYEE\_ROW EMPLOYEE\_CURSOR%ROWTYPE;**

**-- Creating the salary limit exception**

**SALARY\_LIMIT EXCEPTION;**

**-- Marks the start of an executable block**

**BEGIN**

**-- Opening cursor**

**OPEN EMPLOYEE\_CURSOR;**

**FETCH EMPLOYEE\_CURSOR INTO EMPLOYEE\_ROW;**

**-- While employ cursor finds loop**

**WHILE EMPLOYEE\_CURSOR%FOUND LOOP**

**-- If employee salary is less than 20,000 then**

**IF EMPLOYEE\_ROW.JOB\_SALARY < 20000 THEN**

**-- Raise application error**

**RAISE\_APPLICATION\_ERROR(-20001, 'STOP: '|| EMPLOYEE\_ROW.EMPLOYEE\_ROLE\_ID || ' Salary below 20000');**

**END IF;**

**-- If employee salary is over 90,000 then**

**IF EMPLOYEE\_ROW.JOB\_SALARY > 90000 THEN**

**-- Raise salary limits**

**RAISE SALARY\_LIMIT;**

**END IF;**

**-- Feteching the employee cursor into employee row**

**FETCH EMPLOYEE\_CURSOR INTO EMPLOYEE\_ROW;**

**-- Ending the loop**

**END LOOP;**

**-- Closing the employee cursor**

**CLOSE EMPLOYEE\_CURSOR;**

**EXCEPTION**

**WHEN SALARY\_LIMIT THEN**

**-- If any employee has a salary greater than 90,000 output a message**

**DBMS\_OUTPUT.PUT\_LINE('Employee ID: ' || EMPLOYEE\_ROW.EMPLOYEE\_ROLE\_ID|| ' Salary exceeding 90000');**

**-- Re-raise the exception**

**RAISE;**

**-- Marks the end of an executable block**

**END;**

**/ \*/**

**END PETSTORE\_PACKAGE;**

**/**

**Screenshots of procedures and code being used:**

PROCEDURE REMOVELAST\_DISCOUNT;

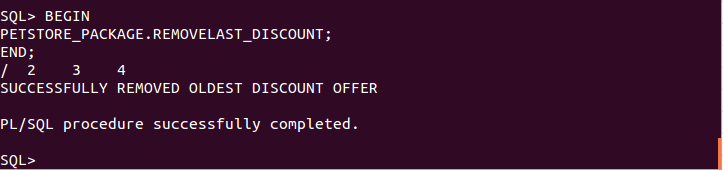
QUERY CODE

**BEGIN**

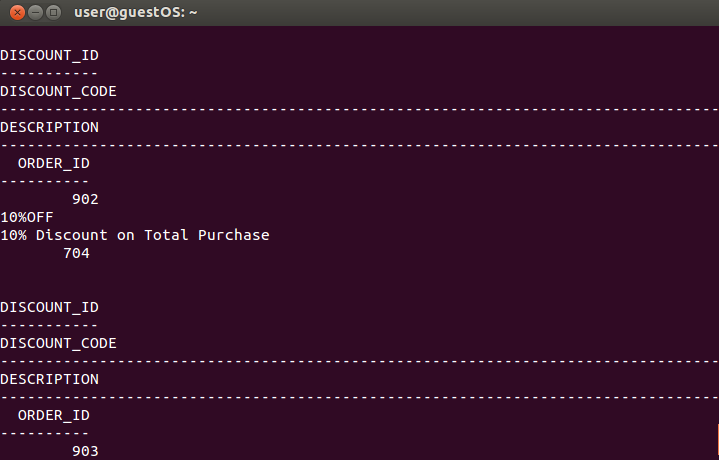
**PETSTORE\_PACKAGE.REMOVELAST\_DISCOUNT;**

**END;**

**/**



We can see here that the “20% off discount” voucher is no longer in the database. The one the manager was suspicious about earlier!



PROCEDURE STOCK\_LOOKUP (INPUT\_ID IN NUMBER);

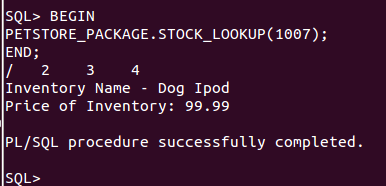
**BEGIN**

**PETSTORE\_PACKAGE.STOCK\_LOOKUP(1007);**

**END;**

**/**

This query shows the item “Dog Ipod” and it’s price.



PROCEDURE COMPARE\_INVENTORY (INPUT1 IN NUMBER, INPUT2 IN NUMBER);

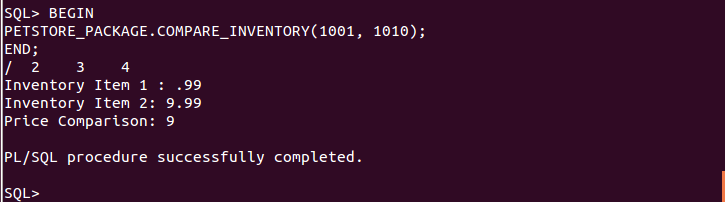
**BEGIN**

**PETSTORE\_PACKAGE.COMPARE\_INVENTORY(1001, 1010);**

**END;**

**/**

This query compares the price of two items and then gives a price comparison.



PROCEDURE SHOW\_EMPLOYEE\_DETAILS (INPUT1 IN NUMBER);

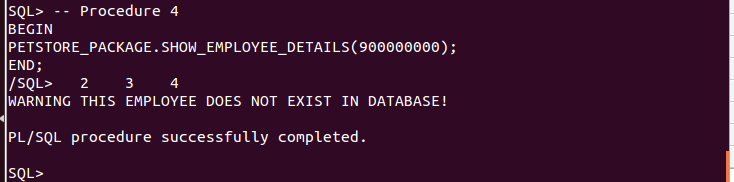
**BEGIN**

**PETSTORE\_PACKAGE.SHOW\_EMPLOYEE\_DETAILS(900000000);**

**END;**

**/**

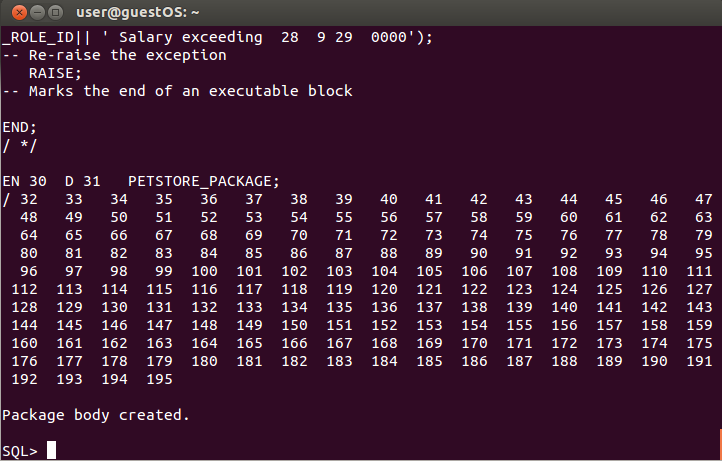
This query shows the employee details when the employee id is used. To test one of the exceptions a wrong value has been entered.



PROCEDURE EMPLOYEE\_SALARY\_RANGE;

This procedure has not been successfully implemented, as it does not pass any information to it. The procedure has been commented on in the main code but requires some further tweaking to work.

**The PETSTORE PACKAGE being successfully created:**



## 2.7. 2 PL/SQL Functions

**PL / SQL FUNCTIONS 1**

This function suggests a pet to buy in the store based on their given budget.

**CREATE OR REPLACE FUNCTION PET\_SUGGESTION (COST IN NUMBER )**

**RETURN VARCHAR2**

**IS**

**ITEM VARCHAR2(50);**

**/\* Set Cursor Item\*/**

**CURSOR ITEM1 IS**

**/\* Select inventory name and where the list price is between 0 and +1500 \*/**

**SELECT INVENTORY\_NAME**

**FROM INVENTORY**

**WHERE LIST\_PRICE BETWEEN COST - 0 AND COST + 1500;**

**/\* Start executing \*/**

**BEGIN**

**OPEN ITEM1;**

**FETCH ITEM1 INTO ITEM ;**

**/\* If no pet in budget \*/**

**IF ITEM1%NOTFOUND THEN**

**DBMS\_OUTPUT.PUT\_LINE('NO PET WITHIN BUDGET RANGE');**

**END IF;**

**CLOSE ITEM1;**

**RETURN ITEM;**

**/\*Error handling message \*/**

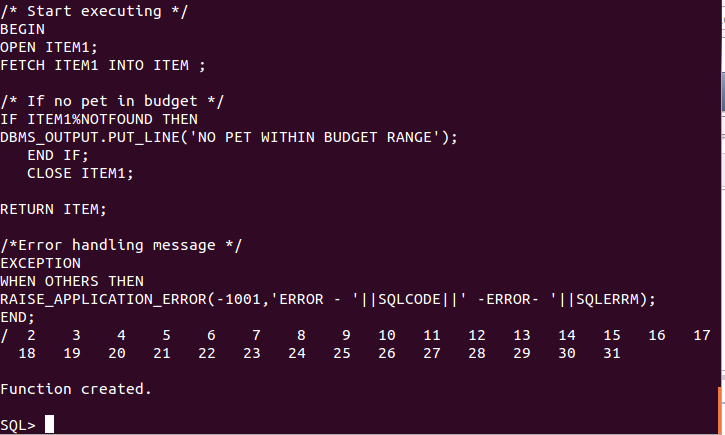
**EXCEPTION**

**WHEN OTHERS THEN**

**RAISE\_APPLICATION\_ERROR(-1001,'ERROR - '||SQLCODE||' -ERROR- '||SQLERRM);**

**END;**

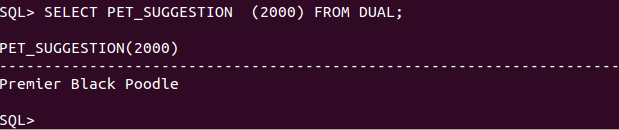
**/**



**Function 1 – TEST**

This select statement will return a pet (dog): Premier Black Poodle

**SELECT PET\_SUGGESTION (2000) FROM DUAL;**



**PL / SQL FUNCTIONS 2**

This function gives the total order amount of an order

**CREATE OR REPLACE FUNCTION TOTAL\_ORDER\_AMOUNT (input\_id IN NUMBER)**

**RETURN NUMBER**

**IS**

**ID\_TO\_USE NUMBER := input\_id;**

**STOCK\_ID NUMBER;**

**TOTAL\_ORDER\_AMOUNT NUMBER := 0;**

**NO\_ORDER\_EXISTS EXCEPTION;**

**BEGIN**

**/\* select sum of order \*/**

**SELECT SUM(I.LIST\_PRICE\*OD.ORDER\_QUANTITY)**

**INTO TOTAL\_ORDER\_AMOUNT**

**FROM ORDER\_DETAILS OD**

**JOIN INVENTORY I**

**ON OD.STOCK\_ID = I.STOCK\_ID**

**WHERE ORDER\_ID = ID\_TO\_USE;**

**/\*If there is no the price i.e raise order doesnt exist \*/**

**IF TOTAL\_ORDER\_AMOUNT IS NULL THEN**

**RAISE NO\_ORDER\_EXISTS;**

**END IF;**

**/\* If there is an order value it will return the total order amount for all orders \*/**

**RETURN TOTAL\_ORDER\_AMOUNT ;**

**EXCEPTION**

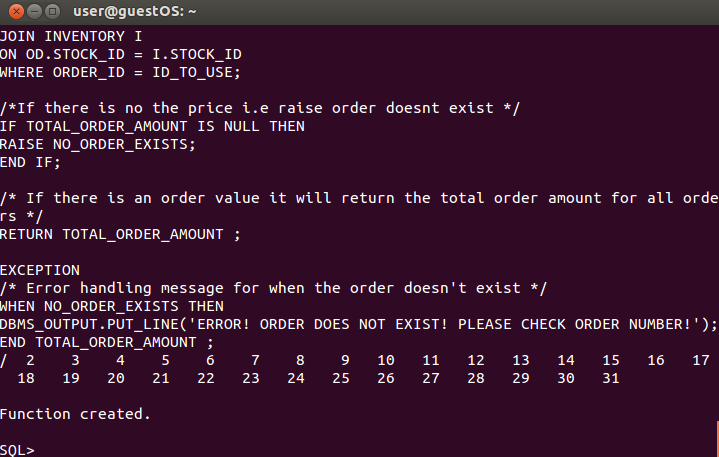
**/\* Error handling message for when the order doesn't exist \*/**

**WHEN NO\_ORDER\_EXISTS THEN**

**DBMS\_OUTPUT.PUT\_LINE('ERROR! ORDER DOES NOT EXIST! PLEASE CHECK ORDER NUMBER!');**

**END TOTAL\_ORDER\_AMOUNT ;**

**/**

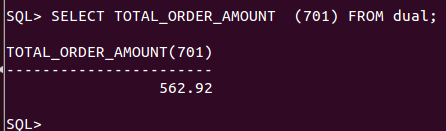


**Function 2 – TEST**

This will calculate the amount 569.92. The price of 7 tennis balls and a Grey tabby.

**SELECT TOTAL\_ORDER\_AMOUNT (701) FROM dual;**

**\*/**



## 2.8. 3 Triggers (at least 1 before, and at least 1 after)

**TRIGGER 1 -** This trigger will automatically calculate the Customer ID with value from a sequence.

**-- Code to create a Sequence**

**CREATE SEQUENCE CUST\_TRIGGER**

**START WITH 513 INCREMENT BY 1;**

**-- Code to create a trigger \*/**

**CREATE OR REPLACE TRIGGER CUST\_TRIGGER**

**-- Trigger to active on customers for each row**

**BEFORE INSERT ON CUSTOMERS FOR EACH ROW**

**BEGIN**

**/\* Select next value from sequence created above \*/**

**SELECT CUST\_TRIGGER.NEXTVAL INTO :NEW.CUSTOMER\_ID FROM DUAL;**

**END;**

**/**

**/\* Demonstrate the trigger working \*/**

**BEGIN**

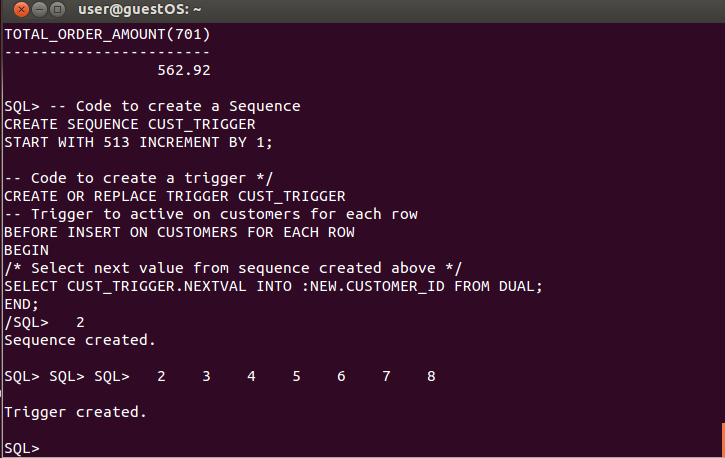
**/\* Insert into Customer \*/**

**INSERT INTO CUSTOMERS (CUST\_FIRST\_NAME, CUST\_LAST\_NAME) VALUES('Bill', 'Murray');**

**COMMIT;**

**END;**

**/**



**Trigger 1 – TEST**

**BEGIN**

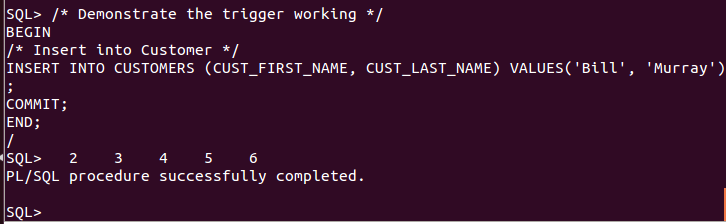
**/\* Insert into Customer \*/**

**INSERT INTO CUSTOMERS (CUST\_FIRST\_NAME, CUST\_LAST\_NAME) VALUES('Bill', 'Murray');**

**COMMIT;**

**END;**

**/**



**Trigger 2 (Before)** - This trigger will output messages when inserting, updating or deleting from the Customers table

**/\* Create trigger \*/**

**CREATE OR REPLACE TRIGGER CUST\_TRIGGER\_OUTPUT**

**/\* Trigger to act before \*/**

**BEFORE INSERT**

**/\* Columns the trigger affects \*/**

**OR UPDATE OF CUST\_PHONE, CUST\_EMAIL**

**OR**

**DELETE**

**/\* Table the trigger is activated on \*/**

**ON CUSTOMERS**

**BEGIN**

**/\*Start case statement \*/**

**CASE**

**/\* When inserting do... \*/**

**WHEN INSERTING THEN**

**/\* Print out statement for insert \*/**

**DBMS\_OUTPUT.PUT\_LINE('NEW DATA HAS SUCCESSFULLY BEEN ENTERED INTO CUSTOMERS');**

**WHEN UPDATING('CUST\_PHONE') THEN**

**/\* Print out statement for update \*/**

**DBMS\_OUTPUT.PUT\_LINE('UPDATING CUSTOMER PHONE NUMBER COMPLETE');**

**/\* When updating do.. \*/**

**WHEN UPDATING('CUST\_EMAIL') THEN**

**/\* Print out statement for update \*/**

**DBMS\_OUTPUT.PUT\_LINE('UPDATING CUSTOMER EMAIL COMPLETE');**

**/\* When deleting do \*/**

**WHEN DELETING THEN**

**/\* Print out statement for delete \*/**

**DBMS\_OUTPUT.PUT\_LINE('DATA HAS BEEN DELETED FROM CUSTOMERS');**

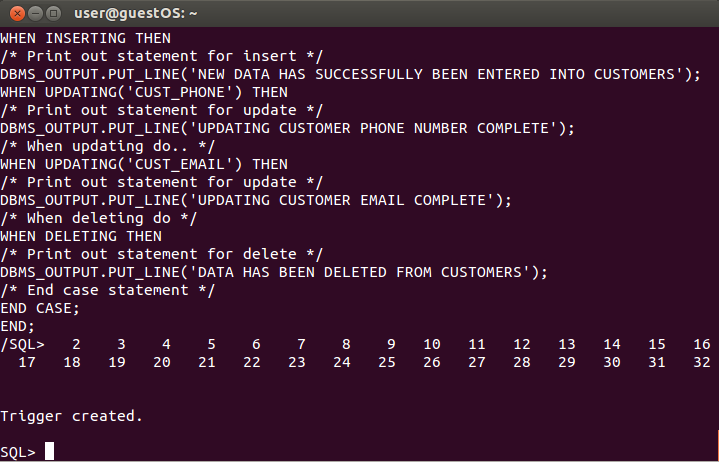
**/\* End case statement \*/**

**END CASE;**

**END;**

**/**

**Creation of the trigger**



**Trigger 2 – TEST**

**BEGIN**

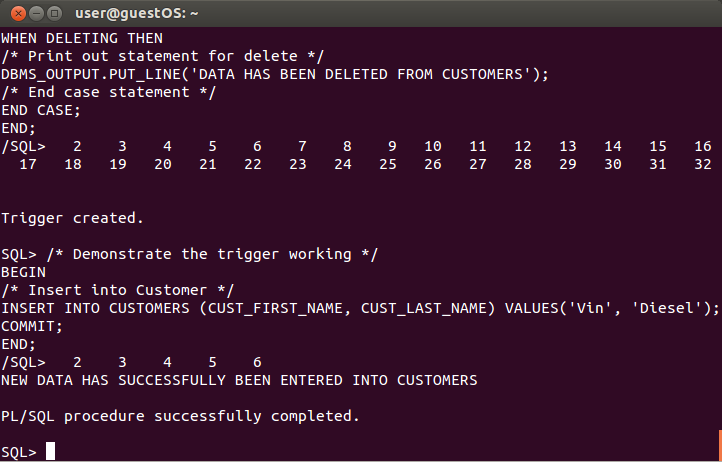
**/\* Insert into Customer \*/**

**INSERT INTO CUSTOMERS (CUST\_FIRST\_NAME, CUST\_LAST\_NAME) VALUES('Vin', 'Diesel');**

**COMMIT;**

**END;**

**/**



Trigger 3

This trigger is to look at the old price and new price of a product, supply or pet changed in the inventory table. This is a vital component for any shop database as it provides insight into the shifts in cost.

**/\* Creates the trigger \*/**

**CREATE OR REPLACE TRIGGER UPDATE\_PRODUCT\_PRICE**

**/\* Specifies the table to insert trigger \*/**

**AFTER DELETE OR INSERT OR UPDATE ON INVENTORY**

**FOR EACH ROW**

**/\* When new list price is greater than 0 this trigger will happen \*/**

**WHEN (NEW.LIST\_PRICE > 0)**

**DECLARE**

**/\* Intilising price difference \*/**

**PRICE\_DIFFERENCE NUMBER;**

**BEGIN**

**/\* Calculating price difference \*/**

**PRICE\_DIFFERENCE := :NEW.LIST\_PRICE - :OLD.LIST\_PRICE;**

**/\* Print statement original price \*/**

**DBMS\_OUTPUT.PUT\_LINE('ORIGINAL PRICE: ' || :OLD.LIST\_PRICE);**

**/\* Print statement updated price \*/**

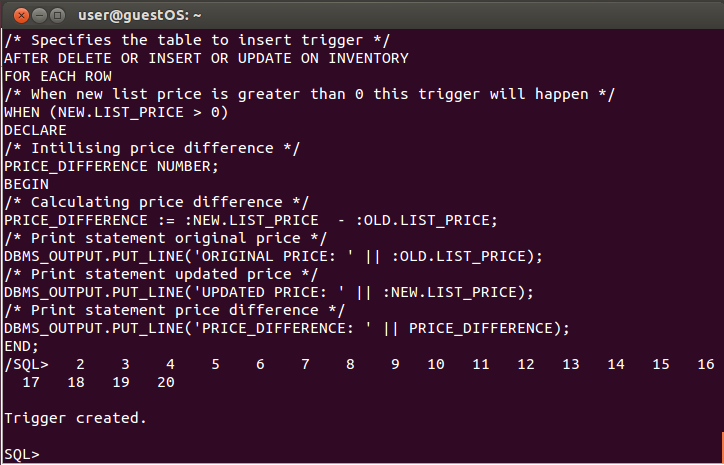
**DBMS\_OUTPUT.PUT\_LINE('UPDATED PRICE: ' || :NEW.LIST\_PRICE);**

**/\* Print statement price difference \*/**

**DBMS\_OUTPUT.PUT\_LINE('PRICE\_DIFFERENCE: ' || PRICE\_DIFFERENCE);**

**END;**

**/**



## 2.9. Identification of weaknesses or potential improvements of the database

There were many outstanding issues during the first few iterations of the database. The main issue being that the order details split into three different categories. This problem has been addressed by implementing the inventory table, which has drastically improved the database. The database could potentially been populated with more data to show the functionally of it better but the core infrastructure as it is transparent and caters for easy data manipulation.

To summarise the **main weaknesses** of the database are:

* The lack of sequences when creating tables. To help save on time entering in data manually i.e hardcoding values in rather than values being automatically generated.
* The lack of constraints preventing the deletion of data.
* The lack of triggers in the database to help ensure data integrity. This was evident especially when implementing the triggers as it showed how powerful they are.

**Potential improvements:**

* **Evaluating the primary key assignments** – I automatically assigned all keys with an ID number. Some of the best databases refine primary keys required for data granularity.
* **Naming Conventions** – I automatically named each column with the table name in mind. This was to ensure that queries were easy to type out but may in fact not actually be best practice.
* **Identifying new attributes** – I was able to add the inventory data from my first design and it gave my database a complete overhaul. I also added the employer role table which improved the flow and queries of the database.
* **Refining attribute atomicity** is a big component that I will examine in the future databases I create. I potentially could have broken down certain components of data collection to get more refined queries.