UCD School of Computer Science and Informatics
Scoil na Ríomheolaíochta agus na Faisnéisíochta UCD

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

PART 2

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.

PART 2

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.

PART 2

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

PART 2

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

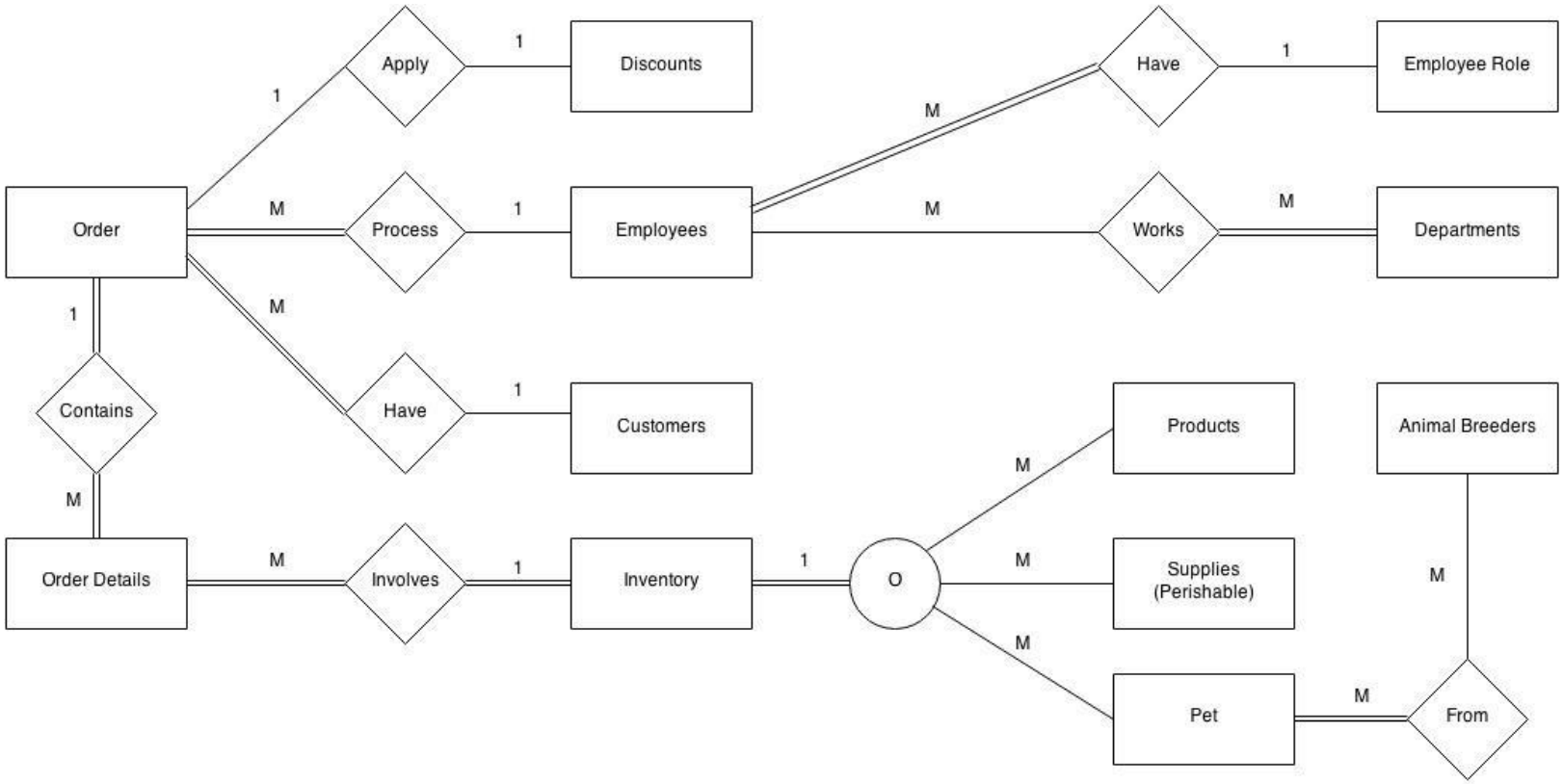
10. One inventory item can involve many pets and many pets are assigned as one inventory item. It is mandatory for inventory to involve pets but optional for pets to be involved in inventory

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.

PART 2

1,5. Final ERD of the Pet Store Database

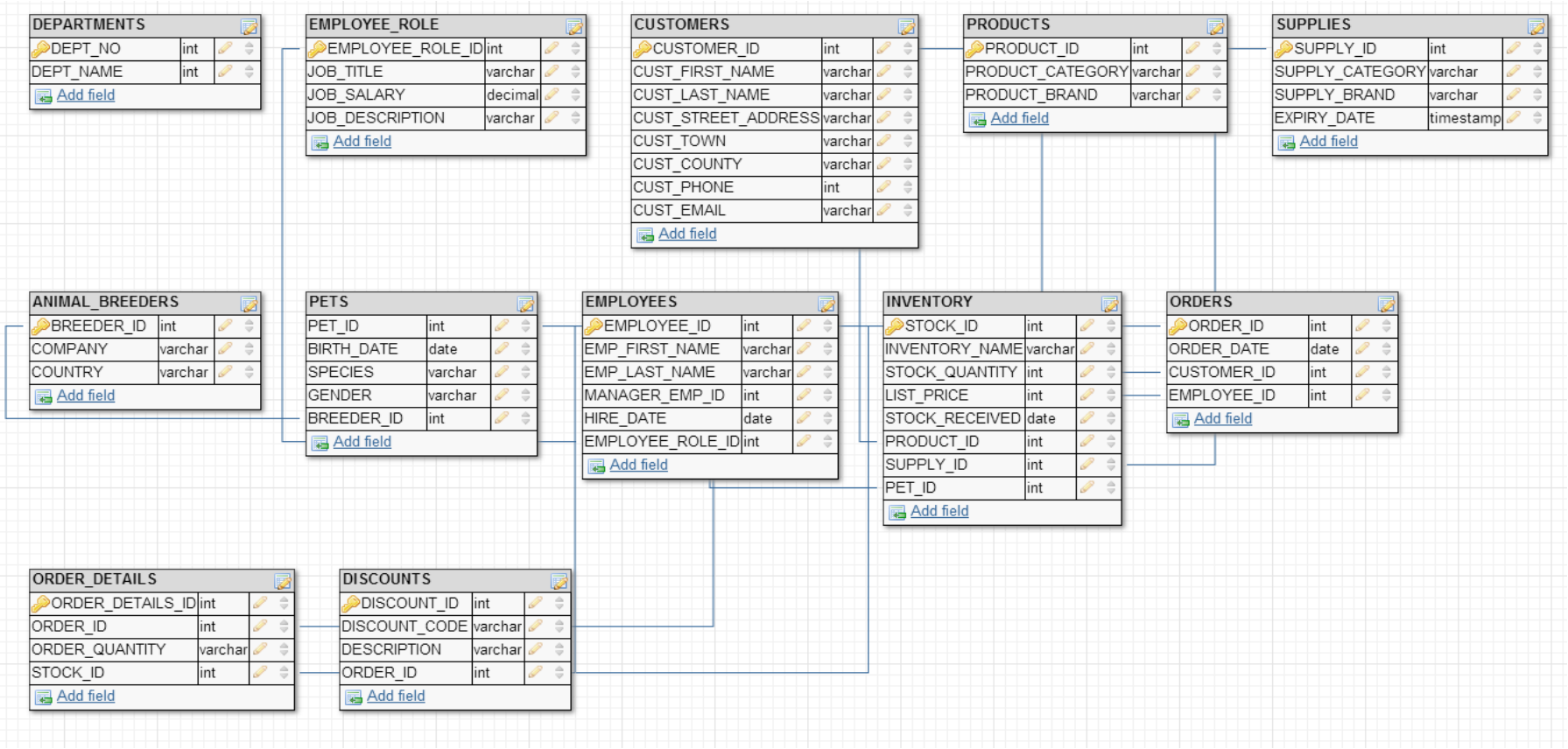
The pet store database has received some improvements from its previous iterations. Most importantly, the issue between order details and products, supplies and pets has been resolved. An inventory table was created to rectify this. Specialisation is also now used to highlight that stock inventory can 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.



PART 2

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)

PART 2

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.

PART 2

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

EMPLOYEE_ID	EMPLOYEE_FULL_NAME	DEPT_NO
610	Donnacha Holmes	5
609	Jonathan Cody	6
608	Siobhain Byrne	9
607	Niamh Hendy	4
606	Shane Herlihy	3
605	Rory Delaney	9
605	Rory Delaney	1
604	Noreen Lenihan	1
604	Noreen Lenihan	2
604	Noreen Lenihan	9
604	Noreen Lenihan	3
603	Ronna Fibikar	9
603	Ronna Fibikar	10
602	Garrett Colemen	10
602	Garrett Colemen	9
601	Zachary Campbell	8
601	Zachary Campbell	6
601	Zachary Campbell	7

18 rows returned in 0.01 seconds Download

PART 2

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

ORDER_ID	STOCK_ID	ORDER_QUANTITY	INVENTORY_NAME	LIST_PRICE	TOTAL_PRICE
708	1015	2	Kitty Cat Bites!	.99	1.98
708	1032	1	Blue-cheeked Amazon	3000.99	3000.99
706	1016	1	Smokey Delights!	.99	.99
705	1018	10	I Cant Believe this is Dog Food!	6.99	69.9
705	1012	2	Meow Yum Yums!	.99	1.98
704	1024	1	Premier White Poodle	2000.99	2000.99
703	1009	2	Red Bowl	9.99	19.98
703	1006	1	KONG Sitting Frog Dog Toy	9.99	9.99
703	1013	1	Barktastic!	2.99	2.99
702	1004	2	Buster Cube	4.99	9.98
702	1007	1	Dog Ipod	99.99	99.99
701	1020	1	Grey Tabby	555.99	555.99
701	1001	7	Tennis Ball	.99	6.93

13 rows returned in 0.02 seconds Download

PART 2

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

EMPLOYEE_FULL_NAM	HIRE_DATE	JOB_TITLE	JOB_SALARY
Zachary Campbell	01/01/2013	Store Manager	90000
Rory Delaney	07/05/2013	Assistant_Manager	50000
Ronna Fibikar	07/22/2013	Assistant_Manager	50000
Garrett Colemen	05/06/2013	Pet_Guru	40000
Noreen Lenihan	03/03/2014	Senior_Sales_Assistant	30000
Donnacha Holmes	02/10/2014	Senior_Sales_Assistant	30000
Jonathan Cody	02/15/2014	Security Guard	25000
Shane Herlihy	12/15/2013	Junior_Sales_Assistant	20000
Niamh Hendy	12/10/2013	Junior_Sales_Assistant	20000
Siobhain Byrne	01/03/2014	Junior_Sales_Assistant	20000

10 rows returned in 0.00 seconds Download

PART 2

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

PET_ID	BIRTH_DATE	SPECIES	GENDER	BREEDER_ID
314	12/13/2013	Parrot	Male	410
313	11/14/2013	Parrot	Male	406
312	10/15/2013	Parrot	Male	408
311	10/15/2013	Parrot	Female	407
310	12/05/2013	Cat	Male	406
309	12/08/2013	Dog	Female	405
308	11/11/2013	Dog	Male	405
307	12/05/2013	Dog	Female	404
306	12/25/2013	Dog	Female	404
305	01/12/2013	Dog	Male	403
304	12/13/2013	Snake	Female	403
303	12/13/2013	Cat	Female	402
302	12/11/2013	Cat	Female	402
301	12/30/2013	Dog	Male	401

14 rows returned in 0.24 seconds Download

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

EMPLOYEE_FULL_NAM	ORDER_DATE	ORDER_ID
Donnacha Holmes	04/08/2014	709
Donnacha Holmes	04/08/2014	710
Garrett Colemen	-	-
Jonathan Cody	-	-
Niamh Hendy	04/05/2014	703
Noreen Lenihan	-	-
Ronna Fibikar	-	-
Rory Delaney	04/05/2014	704
Rory Delaney	04/05/2014	701
Rory Delaney	04/05/2014	705
Shane Herlihy	04/07/2014	702
Siobhain Byrne	04/05/2014	708
Siobhain Byrne	04/05/2014	707
Siobhain Byrne	04/05/2014	706
Zachary Campbell	-	-

15 rows returned in 0.05 seconds Download

Left Outer Join 2 – Description

PART 2

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

PRODUCT_CATEGORY	PRODUCT_BRAND	INVENTORY_NAME	LIST_PRICE	STOCK_ID
Toy	Almasanu	Tennis Ball	.99	1001
Toy	Almasanu	Holee Roller Ball	4.99	1002
Toy	Almasanu	Kong Rubber Toy	3.99	1003
Toy	Almasanu	Buster Cube	4.99	1004
Toy	Almasanu	Omega Paw Tricky Treat Ball	7.99	1005
Toy	Almasanu	KONG Sitting Frog Dog Toy	9.99	1006
Bowls	Bowen	Red Bowl	9.99	1009
Bowls	Bowen	Green Bowl	2.99	1011
Bowls	Bowen	Purple Bowl	9.99	1010
Electronics	Byrne	Dog Ipod	99.99	1007
Electronics	Byrne	Dog Glass	499.99	1008
Id Tags	Casey	-	-	-
Collars	Coleman	-	-	-
Carriers	Harkness	-	-	-
Beds	Knowles	-	-	-
Health	Walsh	-	-	-
Furniture	Walsh	-	-	-
Grooming	Wong	-	-	-

18 rows returned in 0.01 seconds Download

Full Outer Join 1 – Description

PART 2

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

CUSTOMER_ID	CUSTOMER_FULL_NAME	ORDER_ID	ORDER_DATE
501	Andrei Almasanu	701	04/05/2014
501	Andrei Almasanu	702	04/07/2014
503	Aongus Bates	703	04/05/2014
502	Gabriella Bacelli	704	04/05/2014
502	Gabriella Bacelli	705	04/05/2014
507	John Dunnion	706	04/05/2014
505	Ciara Byrne	707	04/05/2014
508	Mike Flynn	708	04/05/2014
504	Laura Duggan	709	04/08/2014
504	Laura Duggan	710	04/08/2014
512	Brian Hassett	-	-
506	David Clarke	-	-
509	Sharon McHugh	-	-
510	Colm Glennon	-	-
511	Desmond Haines	-	-

15 rows returned in 0.01 seconds Download

PART 2

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

DISCOUNT_ID	DISCOUNT_CODE	ORDER_ID
901	20%OFF	701
902	10%OFF	704
903	5%OFF	705
-	-	702
-	-	710
-	-	706
-	-	707
-	-	708
-	-	709
-	-	703

10 rows returned in 0.01 seconds Download

PART 2

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

STOCK_ID	PET_NAME	BIRTH_DATE	SPECIES	GENDER	BREEDER_ID
1032	Blue-cheeked Amazon	12/13/2013	Parrot	Male	407
1031	Blue-cheeked Amazon	11/14/2013	Parrot	Male	410
1030	Cliff Parakeet	10/15/2013	Parrot	Male	410
1029	Cuban Red Macaw	10/15/2013	Parrot	Female	410
1028	Tabby	12/05/2013	Cat	Male	409
1027	Chihuahua	12/08/2013	Dog	Female	401
1026	English Foxhound	11/11/2013	Dog	Male	405
1025	German Shepard	12/05/2013	Dog	Female	405
1024	Premier White Poodle	12/25/2013	Dog	Female	401
1023	Premier Black Poodle	01/12/2013	Dog	Male	402
1022	Python	12/13/2013	Snake	Female	403
1021	White Tabby	12/13/2013	Cat	Female	404
1020	Grey Tabby	12/11/2013	Cat	Female	404
1019	German Shepard	12/30/2013	Dog	Male	402

14 rows returned in 0.02 seconds Download

PART 2

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

ORDER_ID	ORDER_DATE	CUSTOMER_ID	EMPLOYEE_ID	ORDER_DETAILS_ID	ORDER_ID	ORDER_QUANTITY	STOCK_ID
701	04/05/2014	501	605	801	701	7	1001
701	04/05/2014	501	605	802	701	1	1020
702	04/07/2014	501	606	804	702	1	1007
702	04/07/2014	501	606	803	702	2	1004
703	04/05/2014	503	607	806	703	2	1009
703	04/05/2014	503	607	807	703	1	1006
703	04/05/2014	503	607	805	703	1	1013
704	04/05/2014	502	605	808	704	1	1024
705	04/05/2014	502	605	810	705	10	1018
705	04/05/2014	502	605	809	705	2	1012
706	04/05/2014	507	608	811	706	1	1016
708	04/05/2014	508	608	813	708	1	1032
708	04/05/2014	508	608	812	708	2	1015

13 rows returned in 0.01 seconds Download

PART 2

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

ORDER_DETAILS_ID	STOCK_ID	SUM_OF_ALL_ORDERS
801	1001	99
801	-	99
802	1020	555.99
802	-	555.99
803	1004	249.5
803	-	249.5
804	1007	3999.6
804	-	3999.6
805	1013	299
805	-	299
806	1009	399.6
806	-	399.6
807	1006	399.6
807	-	399.6
808	1024	2000.99
808	-	2000.99
809	1012	99
809	-	99
810	1018	279.6
810	-	279.6
-	1001	99
-	1004	249.5
-	1006	399.6
-	1007	3999.6
-	1009	399.6
-	1012	99
-	1013	299
-	1015	99
-	1016	99
-	1018	279.6
-	1020	555.99
-	1024	2000.99
-	1032	3000.99
-	-	11580.87

40 rows returned in 0.10 seconds Download

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

CUSTOMER_ID	CUSTOMER_FULL_NAME
501	Andrei Almasanu
505	Ciara Byrne
506	David Clarke
504	Laura Duggan
508	Mike Flynn
509	Sharon McHugh

6 rows returned in 0.02 seconds Download

PART 2

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

SPECIES	GENDER	COUNT(*)
Cat	Female	2
Dog	Female	3

PART 2

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

DISCOUNT_ID	DISCOUNT_CODE	DESCRIPTION	ORDER_ID
901	20%OFF	20% Discount on Total Purchase	701

1 rows returned in 0.01 seconds Download

PART 2

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

STOCK_ID	INVENTORY_NAME	LIST_PRICE	PRODUCT_ID	SUPPLY_ID	PET_ID
1032	Blue-cheeked Amazon	3000.99	-	-	314

1 rows returned in 0.01 seconds Download

PART 2

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

EMPLOYEE_ID	EMP_FIRST_NAME	EMP_LAST_NAME	JOB_SALARY	JOB_TITLE
606	Shane	Herlihy	20000	Junior_Sales_Assistant
607	Niamh	Hendy	20000	Junior_Sales_Assistant
608	Siobhain	Byrne	20000	Junior_Sales_Assistant

3 rows returned in 0.01 seconds Download

PART 2

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;
/
```

PART 2

Screenshots of procedures and code being used:

PROCEDURE REMOVELAST_DISCOUNT;

QUERY CODE

```
BEGIN
PETSTORE_PACKAGE.REMOVELAST_DISCOUNT;
END;
/
```

```
SQL> BEGIN
PETSTORE_PACKAGE.REMOVELAST_DISCOUNT;
END;
/  2     3     4
SUCCESSFULLY REMOVED OLDEST DISCOUNT OFFER

PL/SQL procedure successfully completed.

SQL>
```

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

```
user@guestOS: ~

DISCOUNT_ID
-----------
DISCOUNT_CODE
--------------------------------------------------------------------------------
DESCRIPTION
--------------------------------------------------------------------------------
  ORDER_ID
----------
        902
10%OFF
10% Discount on Total Purchase
       704


DISCOUNT_ID
-----------
DISCOUNT_CODE
--------------------------------------------------------------------------------
DESCRIPTION
--------------------------------------------------------------------------------
  ORDER_ID
----------
        903
```

PART 2

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.

```
SQL> BEGIN
PETSTORE_PACKAGE.STOCK_LOOKUP(1007);
END;
/   2    3    4
Inventory Name - Dog Ipod
Price of Inventory: 99.99

PL/SQL procedure successfully completed.

SQL>
```

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.

```
SQL> BEGIN
PETSTORE_PACKAGE.COMPARE_INVENTORY(1001, 1010);
END;
/  2    3    4
Inventory Item 1 : .99
Inventory Item 2: 9.99
Price Comparison: 9

PL/SQL procedure successfully completed.

SQL>
```

PART 2

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.

```
SQL> -- Procedure 4
BEGIN
PETSTORE_PACKAGE.SHOW_EMPLOYEE_DETAILS(900000000);
END;
/SQL>   2    3    4
WARNING THIS EMPLOYEE DOES NOT EXIST IN DATABASE!

PL/SQL procedure successfully completed.

SQL>
```

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:

```
user@guestOS: ~
_ROLE_ID|| ' Salary exceeding  28  9 29  0000');
-- Re-raise the exception
   RAISE;
-- Marks the end of an executable block

END;
/ */

EN 30  D 31   PETSTORE_PACKAGE;
/ 32   33   34   35   36   37   38   39   40   41   42   43   44   45   46   47
  48   49   50   51   52   53   54   55   56   57   58   59   60   61   62   63
  64   65   66   67   68   69   70   71   72   73   74   75   76   77   78   79
  80   81   82   83   84   85   86   87   88   89   90   91   92   93   94   95
  96   97   98   99  100  101  102  103  104  105  106  107  108  109  110  111
 112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127
 128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143
 144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159
 160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175
 176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191
 192  193  194  195

Package body created.

SQL>
```

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;
/
```

```
/* 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;
/  2    3    4    5    6    7    8    9   10   11   12   13   14   15   16   17
  18   19   20   21   22   23   24   25   26   27   28   29   30   31

Function created.

SQL>
```

PART 2

Function 1 – TEST

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

```
SELECT PET_SUGGESTION (2000) FROM DUAL;
```

```
SQL> SELECT PET_SUGGESTION  (2000) FROM DUAL;

PET_SUGGESTION(2000)
--------------------------------------------------------------------
Premier Black Poodle

SQL>
```

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 ;
/
```

PART 2

```
user@guestOS: ~
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 orde
rs */
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 ;
/  2    3    4    5    6    7    8    9   10   11   12   13   14   15   16   17
  18   19   20   21   22   23   24   25   26   27   28   29   30   31

Function created.

SQL>
```

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;
***/**

```
SQL> SELECT TOTAL_ORDER_AMOUNT  (701) FROM dual;

TOTAL_ORDER_AMOUNT(701)
-----------------------
                 562.92

SQL>
```

PART 2

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;
/
```

```
user@guestOS: ~
TOTAL_ORDER_AMOUNT(701)
-----------------------
                  562.92

SQL> -- 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;
/SQL>   2
Sequence created.

SQL> SQL> SQL>   2    3    4    5    6    7    8

Trigger created.

SQL>
```

PART 2

Trigger 1 – TEST

```
BEGIN
/* Insert into Customer */
INSERT INTO CUSTOMERS (CUST_FIRST_NAME, CUST_LAST_NAME) VALUES('Bill', 'Murray');
COMMIT;
END;
/
```

```
SQL> /* Demonstrate the trigger working */
BEGIN
/* Insert into Customer */
INSERT INTO CUSTOMERS (CUST_FIRST_NAME, CUST_LAST_NAME) VALUES('Bill', 'Murray')
;
COMMIT;
END;
/
SQL>   2    3    4    5    6
PL/SQL procedure successfully completed.

SQL>
```

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

```
user@guestOS: ~
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;
/SQL>   2    3    4    5    6    7    8    9   10   11   12   13   14   15   16
  17   18   19   20   21   22   23   24   25   26   27   28   29   30   31   32


Trigger created.

SQL>
```

Trigger 2 – TEST

```
BEGIN
/* Insert into Customer */
INSERT INTO CUSTOMERS (CUST_FIRST_NAME, CUST_LAST_NAME) VALUES('Vin', 'Diesel');
COMMIT;
END;
/
```

```
user@guestOS: ~
WHEN DELETING THEN
/* Print out statement for delete */
DBMS_OUTPUT.PUT_LINE('DATA HAS BEEN DELETED FROM CUSTOMERS');
/* End case statement */
END CASE;
END;
/SQL>   2    3    4    5    6    7    8    9   10   11   12   13   14   15   16
  17   18   19   20   21   22   23   24   25   26   27   28   29   30   31   32


Trigger created.

SQL> /* Demonstrate the trigger working */
BEGIN
/* Insert into Customer */
INSERT INTO CUSTOMERS (CUST_FIRST_NAME, CUST_LAST_NAME) VALUES('Vin', 'Diesel');
COMMIT;
END;
/SQL>   2    3    4    5    6
NEW DATA HAS SUCCESSFULLY BEEN ENTERED INTO CUSTOMERS

PL/SQL procedure successfully completed.

SQL>
```

Trigger 3

PART 2

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;
/
```

```
user@guestOS: ~
/* 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;
/SQL>   2    3    4    5    6    7    8    9   10   11   12   13   14   15   16
  17   18   19   20

Trigger created.

SQL>
```

PART 2

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.