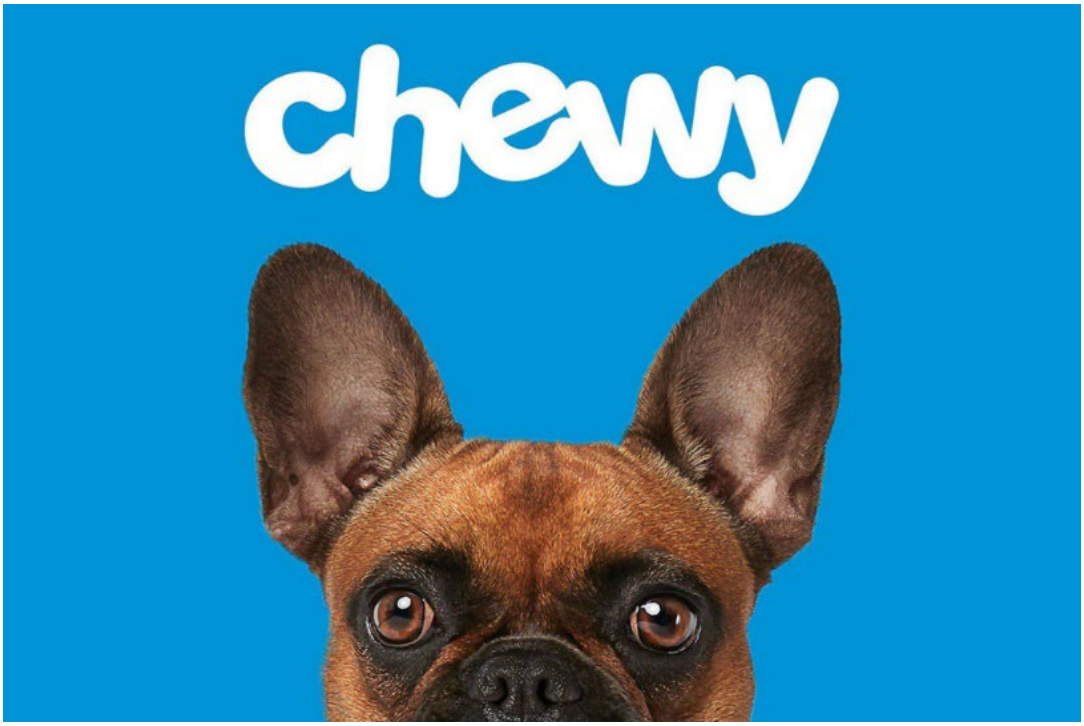


MIS 531 Enterprise Data Management

Eller's EDMers

1. Jaimin Fojdar
2. Medha Singh
3. Maximillian Goswitz
4. Unnati Palande
5. Bhargavi S Muralidhara
6. Tanishka Shorey

December 7th, 2022

Table of Contents

Chapter 1: Requirement Analysis

Chewy is an online retailer that serves as a "one stop shop" for all pet supplies. Despite the success of their firm, they have encountered several problems. The various entities and operations that they collaborate with must be supported by the Chewy database. The relevant information is described below.

Chewy allows customers to place orders. Chewy keeps a distinct customer ID, first and last name, username, password, phone number, email address, and residential address for each customer (street, city, state, and ZIP). On Chewy, a consumer may place numerous orders, but each order can only have one customer as its owner.

A client has a pet. No pet is required for a Chewy customer to be one, but each pet in the company's database can only belong to one customer. A pet's ID, name, breed, weight, adoption date, birthday, age, and kind (dog, cat, or bird) are among the details that are saved about them.

Many people work for Chewy. Chewy keeps a record of each employee's employee ID, first, middle, and last name, email, home address (street, city, state, and ZIP), SSN, home and work phone numbers, and the date the employee was hired. Each employee's login must be created with a special username and password.

There are just six distinct job categories at Chewy: adoption/rescue associates, managers, insurance agents, developers, veterinarians, and customer support.

We keep track of a veterinarian's credentials (which may be numerous), their area of expertise (such as dental hygiene or physical examinations), and the clinic location (street, city, state, and ZIP) where they practice.

For developers, we want to know how many coding languages they are familiar with, their hourly rate, and their area of expertise (i.e., front end, back end, etc.). One or more training courses are also created by developers. We keep track of the course ID, course title, start date, and course length for training courses.

Employees in customer service are identified by their location, the states they serve, and the languages they speak.

Finally, the database stores the department, years of management, and highest degree attained for managers. An employee can only be assigned to one manager at a time, and managers must oversee at least one employee (but employees may not be assigned to a manager).

Insurance agents are responsible for providing insurance and are required to have a license, certificate, and years of experience. In addition to the policy ID and other information, insurance also includes the premium amount, start date, policy summary, policy name, and due date. Although an insurance policy may not have yet been issued, an insurance agent must offer at

least one insurance policy in order to practice as an insurance agent. Numerous insurance agents can also offer an insurance policy.

Adoption/Rescue Associates' availability (full- or part-time) and shift schedules are detailed in the position description. A local shelter may have zero or many associates working for them, and an adoption/rescue associate may work with many shelters. Shelter ID, shelter name, shelter address (street, city, state, and ZIP), point of contact name, point of contact email, and point of contact number exist for local shelters.

Chewy clients place orders that result in an invoice. The payment method, the status, the address (street, city, state, and ZIP), and the invoice ID are all included in each invoice. An order creates at least one invoice, and every invoice is associated with a single order.

Orders may include both goods and veterinary services. Every order that contains items generates item details, which specify the item quantity within a given order. The following information can be used to track veterinary services: service type, order date, location, and service ID. You can track the details of your services. We can save the service duration for each veterinary service, which can be determined using the start and end times of the service. Veterinarians provide services to animals. Services of all kinds, including parasite management, immunization, grooming, and dental, are available. Type of parasite, medication provided, and dosage all factor towards parasite control. It is possible to track vaccination information such as the date of the most recent vaccine, the date of the next vaccination, and the pet's next vaccination ID. This immunization is just one of many that are offered at Chewy's locations. The vaccination ID and vaccination name are tracked by the clinics. Grooming information can be stored, including the materials used during the session and the type of grooming, including haircut, bathing, and nail trimming. We can track dental information such as procedures performed, medications given to pets, and equipment used.

Chewy's portfolio includes a wide range of goods. The different items' product types should be stored. Each product type has an ID and description. Additionally, each item should contain its ID, brand, price, discount price, description, and the type of pet it is intended for. Each item is either a retail or pharmaceutical product. Veterinarians must prescribe pharmacy items, which include information on the type of treatment, dose frequency, and dosage amount daily. For the purpose of purchasing, we keep records of the supply category, which identifies a product's category depending on the kind or breed of the pet. Stockpiles contain items that are kept. Inventory includes details such as inventory ID and inventory name, which is just the name of the fulfillment center. We must keep track of and update the number of each item that is added to the inventory.

A vendor's ability to furnish only certain product kinds limits the range of goods that they can offer. A product may be provided by several vendors. Every vendor type has a vendor type ID and a vendor type name, and there are different vendor types. Basic vendor information including ID,

company name, contact name, contact number, contact email, and vendor address (street, city, state, and ZIP) must be kept on file.

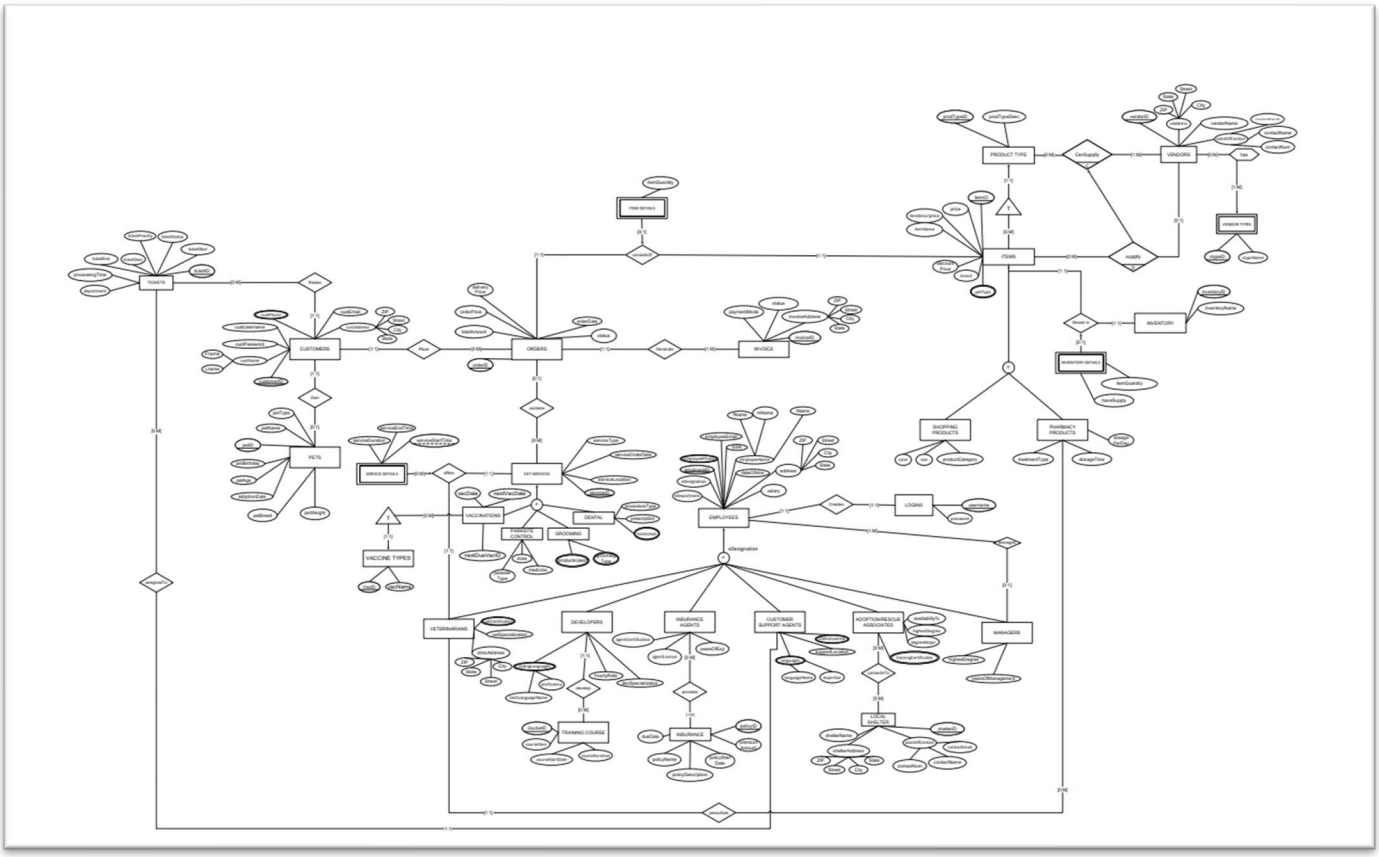
Customers may submit tickets through the system. Each ticket has a unique ID number, description, priority, start date, finish date, status, ticket department, and processing time (can be derived from ticket start and end time). On these tickets, workers are employed. Only one customer may be associated with a ticket, and that same customer may raise many tickets. Although an employee is usually given a ticket, they could not be handling any at a moment.

Chapter 2: ER Diagram and Data Dictionary

Entity Relation Diagram

Visio file:

Eller EDMer's ERD 12-5-2022.vsdx



Data Dictionary

Schema Construct	Construct Description	Other Information
CUSTOMERS		
custAddress	Customer's home address	Composite Attribute: ZIP+Street+City+State
custEmail	Customer Email address	
<u>customerID</u>	Unique ID given to each customer	*Identifying Attribute*
custName	Customer's name	Composite Attribute: fName+lName
custPassword	Customer's password to login	
custPhone	Customer's phone number	Multivalued Attribute
custUsername	Customer's username to login	UNIQUE attribute

Schema Construct	Construct Description	Other Information
EMPLOYEES		
address	Employee home address	Composite Attribute: ZIP+Street+City+State
dateOfHire	When an employee was hired by the company	<u>Date Format</u>: YYYY-MM-DD
eDesignation	An employee's designation	Corresponds with subclasses. CHECK constraint applied, must be in ('VETERINARIANS', 'DEVELOPERS', 'INSURANCE_AGENTS', 'CUSTOMER_SUPPORT_AGENTS', 'MANAGERS').

eDepartment	The department an employee is assigned to	
employeeEmail	Employee email address	
<u>employeeID</u>	Unique employee ID	*Identifying Attribute*
employeeName	Employee name	Composite Attribute: fName+ mName+ lName
employeePhone	Employee phone #	Multivalued Attribute
salary	Yearly salary	
SSN	Social security number for employee	UNIQUE attribute
SUBCLASS #1		
ADOPTION/RESCUE ASSOCIATE	Subclass of EMPLOYEES	
availability%	% time of availability; can be a full time (1) or part-time (0.5 or 0.25 etc.)	should be <=1
degreeMajor	The major of a degree received by an adoption associate	
highestDegree	The highest degree earned by an adoption associate	
trainingCertificates	The type of training certificate received by an adoption associate	Multivalued attribute
SUBCLASS #2		
CUSTOMER SUPPORT AGENTS	Subclass of EMPLOYEES	
languages	The languages known/spoken by an employee	Multivalued Composite Attribute: languageName and expertise
statesCovered	The states a customer support employee takes calls from	Multivalued Attribute
supportLocation	The office name where a customer support employee is located	

SUBCLASS #3		
DEVELOPERS	Subclass of EMPLOYEES	
codingLanguages	Number of coding languages known	Multivalued Composite Attribute: techLanguageName and proficiency
devSpecialization	Specific area in which an employee works (i.e., Front End, Back End, etc.)	
hourlyRate	Current pay rate for a developer	
SUBCLASS #4		
INSURANCE AGENTS	Subclass of EMPLOYEES	
agentCertification	Certification details of agent	
agentLicense	Agent's License #	
yearsOfExp	Agent's years of experience	
SUBCLASS #5		
MANAGERS	Subclass of EMPLOYEES	
highestDegree	Highest degree earned by a manager	
yearsOfManagement	Number of years a manager has been managing	
SUBCLASS #6		
VETERINARIANS	Subclass of EMPLOYEES	
clinicAddress	Address of the vet's clinic	Composite Attribute: ZIP+Street+City+State
vetCertificates	Certificate required to treat different kinds of pets (i.e. special certificate to treat reptiles or horses)	Multivalued Attribute: a vet can be trained to treat more than one kind of animal
vetSpecialization	Which animal a vet specializes in treating	

Schema Construct	Construct Description	Other Information
INVENTORY		
<u>inventoryID</u>	Inventory Identifier	*Identifying Attribute*
inventoryName	Name of an inventory center	

Schema Construct	Construct Description	Other Information
INVENTORY DETAILS	Weak entity class derived from the relationship between INVENTORY & ITEMS	
haveSupply	If inventory has supply for a product	1 = Yes, 0 = No Derived attribute
itemQuantity	How much of a particular item is stored in an inventory	

Schema Construct	Construct Description	Other Information
INVOICE		
invoiceAddress	Address details	Composite Attribute: ZIP+Street+City+State
<u>invoiceID</u>	Invoice identifier	*Identifying Attribute*
paymentMode	Mode of the payment made	
status	Status of payment	

Schema Construct	Construct Description	Other Information
INSURANCE		
dueDate	Due date of the policy	Date Format: YYYY-MM-DD
policyDescription	Description of policy	

<u>policyID</u>	Insurance policy Identifier	*Identifying Attribute*
policyName	Name of the policy	
policyStartDate	Date when the policy is started	Date Format: YYYY-MM-DD
premiumAmount	Premium Amount of policy	

Schema Construct	Construct Description	Other Information
ITEMS	ITEMS is a typing instance of PRODUCT TYPE	
brand	Brand (company) of the item	
discountPrice	The discounted price of an item	
itemDescription	Description of particular food/toy	
<u>itemID</u>	Unique item ID	*Identifying Attribute*
itemName	The name of an item	
petType	Pet type an item was created for	Multivalued Attribute: an item can be created for more than one pet
price	Price of the item	
SUBCLASS #1		
SHOPPING PRODUCTS	Subclass of ITEMS	
color	Color of a shopping product	
productCategory	Category of shopping products for different breeds	
size	Size in number (like pet clothes) for an item	
SUBCLASS #2		
PHARMACY PRODUCTS	Subclass of ITEMS	
dosagePerDay	Dosage intake per day	

dosageTime	Duration (in days) for taking medicines	
treatmentType	Type of treatment needed for pet	

Schema Construct	Construct Description	Other Information
ITEM DETAILS	Weak entity class derived from the relationship between ORDERS & ITEMS	
itemQuantity	The quantity of a particular item within an order	

Schema Construct	Construct Description	Other Information
LOCAL SHELTER		
shelterAddress	Address of the Shelter	Composite Attribute: ZIP+Street+City+State
pointOfContact	Point of Contact for a shelter	Composite Attribute: contactName, contactNum, contactEmail
<u>shelterID</u>	The ID assigned to a particular shelter	*Identifying Attribute*
shelterName	Name of a shelter	

Schema Construct	Construct Description	Other Information
LOGINS		
username	The username an employee will log in with	*Identifying Attribute*
password	The password associated with a username	

Schema Construct	Construct Description	Other Information
ORDERS		
deliveryPrice	Price associated with an order delivery	
orderDate	Date an order was placed	DateTime Format: YYYY-MM-DD hh:mm:ss
<u>orderID</u>	Unique ID for each order	*Identifying Attribute*
orderPrice	The price of all items contained in an order	Derived attribute from price (in ITEMS) * itemQuantity (in ITEM DETAILS)
status	Status of the Order	
totalAmount	The total order amount	Derived Attribute from orderPrice + deliveryPrice

Schema Construct	Construct Description	Other Information
PRODUCT TYPE	Typing Class: ITEMS is the instantiation class.	
prodTypeDesc	Description of product type	
<u>prodTypeID</u>	Unique product type ID	*Identifying Attribute*

Schema Construct	Construct Description	Other Information
TRAINING COURSE		
courseDesc	Description of a course	
courseDuration	How long a course will last	Time Format (hh:mm:ss).
<u>courseID</u>	Unique ID associated with a course	*Identifying Attribute*

courseStartDate	The day a course will begin	

Schema Construct	Construct Description	Other Information
VET SERVICES		
<u>serviceID</u>	Unique service ID	*Identifying Attribute*
serviceLocation	Name of service center	
serviceOrderDate	Date of booking the service	DateTime Format: YYYY-MM-DD hh:mm:ss
serviceType	Type of service	Corresponds with subclasses. CHECK constraint applied, must be in (‘VACCINATIONS’, ‘PARASITE_CONTROL’, ‘GROOMING’, ‘DENTAL’).
SUBCLASS #1		
PARASITE CONTROL	Subclass of VET SERVICES	
dose	How much of a dose is to be taken (daily)	
medicine	Medicine prescribed	
parasiteType	Type of parasite	
SUBCLASS #2		
VACCINATIONS	Subclass of VET SERVICES	
nextDueVacID	The next vaccine ID to be received	
nextVacDate	Due date for the next vaccination	Date Format: YYYY-MM-DD
vacDate	The date when a vaccine was received	Date Format: YYYY-MM-DD
SUBCLASS #3		

GROOMING	Subclass of VET SERVICES	
groomingType	Which grooming services are being provided	Multivalued attribute
productsUsed	What products are used for the chosen service	Multivalued attribute
SUBCLASS #4		
DENTAL	Subclass of VET SERVICES	
prescription	Prescription details after consultation	
procedureType	Type of procedure performed	
toolsUsed	Tools used for the dental procedure	Multivalued attribute

Schema Construct	Construct Description	Other Information
PETS		
adoptionDate	Date of adopting the pet	Date Format: YYYY-MM-DD
petAge	Pet age	Derived attribute from today's date - petBirthday (in years).
petBirthday	Pet Birth Date	
petBreed	Breed of the Pet	
<u>petID</u>	Pet Identifier	*Identifying Attribute*
petName	Pet Name	
petType	Type of Pet – Cat, dog, bird, etc	
petWeight	Weight of the pet	

Schema Construct	Construct Description	Other Information

SERVICE DETAILS	Weak entity class derived from the relationship between VETERINARIANS & VET SERVICES	
serviceDuration	Time taken for service	Derived attribute from serviceEndTime - serviceStartTime. Time Format (hh:mm:ss).
serviceStartTime	Time of starting the service	DateTime Format: YYYY-MM-DD hh:mm:ss. *Partial Identifier*.
serviceEndTime	Time of service completion	DateTime Format: YYYY-MM-DD hh:mm:ss

Schema Construct	Construct Description	Other Information
TICKETS		
department	What department a ticket is assigned to be handled by	
processingTime	Total duration of processing the ticket	Derived attribute from ticketEnd-ticketStart. Time Format (hh:mm:ss).
ticketDesc	Description of the ticket	
ticketEnd	Timestamp of ticket closed	DateTime Format: YYYY-MM-DD hh:mm:ss
<u>ticketID</u>	Unique ticket number	*Identifying Attribute*
ticketPriority	Priority of ticket – Low, Medium, or High.	Check constraint applied, must be in one of the three categories.
ticketStart	Timestamp of ticket raised	DateTime Format: YYYY-MM-DD hh:mm:ss.

ticketStatus	Status of the ticket – open, WIP, Closed, delayed.	

Schema Construct	Construct Description	Other Information
VACCINE TYPES	Typing Class. VACCINATIONS is the instantiation class.	
<u>vacID</u>	The ID associated with a particular vaccine type	*Identifying attribute*
vacName	The name of a Vaccine Type (EX: Rabies for Dogs, Diabetes, etc.)	

Schema Construct	Construct Description	Other Information
VENDORS		
pointOfContact	Point of Contact for a vendor	Composite Attribute: contactName, contactNum, contactEmail
vAddress	Vendor's address	Composite Attribute: ZIP+Street+City+State
<u>vendorID</u>	Unique vendor identifier	*Identifying Attribute*
vendorName	The name of a vendor	

Schema Construct	Construct Description	Other Information
VENDOR TYPES	Composite Entity.	
<u>vTypeID</u>	Unique vendor type identifier	*Identifying Attribute*
vTypeName	Department that vendor caters to	

Relationship Translations

Schema Construct	Construct Description
AssignedTo	Relationship between TICKETS & CUSTOMER SUPPORT AGENTS
CanSupply	Relationship between PRODUCT TYPE & VENDORS
Connects to	Relationship between ADOPTION ASSOCIATE & LOCAL SHELTER.
ConsistsOf	Relationship between ORDERS & ITEMS. ITEM DETAILS weak entity class comes from this relationship.
Contains	Relationship between ORDERS & VET SERVICES
Creates	Relationship between EMPLOYEES & LOGINS.
Develop	Relationship between DEVELOPERS & TRAINING COURSE.
Generate	Relationship between ORDERS & INVOICE
Has	Relationship between VENDORS & VENDOR TYPES.
Manages	Relationship between EMPLOYEES & MANAGERS.
Offers	Relationship between VET SERVICES & VETERINARIANS. SERVICE DETAILS as a weak entity class comes from this relationship.
Own	Relationship between CUSTOMERS & PETS
Place	Relationship between CUSTOMERS & ORDERS

Prescribes	Relationship between PHARMACY PRODUCTS & VETERINARIANS.
Provides	Relationship between INSURANCE AGENTS & INSURANCE.
Raises	Relationship between CUSTOMERS & TICKETS
Stored In	Relationship between INVENTORY & ITEMS. INVENTORY DETAILS weak entity class comes from this relationship.
Supply	Relationship between VENDORS & ITEMS

Chapter 3: Relational Schema

ER to Relational

STRONG ENTITIES (subclasses & multi-valued attributes listed underneath their parent class)

CUSTOMERS (customerID, Fname, Lname, custUsername, custPassword, custEmail, State, Street, City, ZIP)

- UNIQUE constraint on custUsername.

CUSTOMER_PHONES (customerID, custPhone)

- *FOREIGN KEY* (customerID) References **CUSTOMERS** (customerID)

EMPLOYEES (employeeID, fName, mName, lName, eDesignation, eDepartment, employeeEmail, salary, SSN, dateOfHire, State, Street, City, ZIP, managerID)

- UNIQUE Constraint on SSN.
- *FOREIGN KEY* (employeeID) References to **MANAGERS** (employeeID)

EMPLOYEE_PHONES (employeeID, employeePhone)

- *FOREIGN KEY* (employeeID) References **EMPLOYEES** (employeeID)

EMPLOYEE SUBCLASSES

ADOTPTION_ASSOCIATES (employeeID, highestDegree, degreeMajor, availabiltyPercentage)

- *FOREIGN KEY* (employeeID) References **EMPLOYEES** (employeeID)

ADOPTION_ASSOCIATE_TRAINING (employeeID, trainingCertificateID)

- *FOREIGN KEY* (employeeID) References **ADOTPTION_ASSOCIATES** (employeeID)

CUSTOMER_SUPPORT_AGENTS (employeeID, supportLocation)

- *FOREIGN KEY* (employeeID) References **EMPLOYEES** (employeeID)

CUST_SUPPORT_AGENTS_LANGUAGES (employeeID, languageName, expertise)

- *FOREIGN KEY* (employeeID) References **CUSTOMER_SUPPORT** (employeeID)

CUST_SUPPORT_AGENTS_STATES (employeeID, stateCovered)

- *FOREIGN KEY* (employeeID) References **CUSTOMER_SUPPORT** (employeeID)

DEVELOPERS (employeeID, hourlyRate, devSpecialization)

- *FOREIGN KEY* (employeeID) References **EMPLOYEES** (employeeID)

DEVELOPER_CODING (employeeID, techLanguageName, proficiency)

- *FOREIGN KEY* (employeeID) References **DEVELOPERS** (employeeID)

MANAGERS (employeeID, highestDegree, yearsOfManagement)

- *FOREIGN KEY* (employeeID) References to **EMPLOYEES** (employeeID)
- CHECK constraint highestDegreee IN ("Masters", "Doctoral")

INSURANCE_AGENTS (employeeID, agentCertification, agentLicense, yearsOfExp)

- *FOREIGN KEY* (employeeID) References **EMPLOYEES** (employeeID)

VETERINARIANS (employeeID, vetSpecialization, State, Street, City, ZIP)

- *FOREIGN KEY* (employeeID) References **EMPLOYEES** (employeeID)

VETERINARIANS_CERTIFICATES (employeeID, vetCertificateID)

- *FOREIGN KEY* (employeeID) References **VETERINARIANS** (employeeID)

END OF EMPLOYEE SUBCLASSES

INSURANCE (policyID, premiumAmount, policyStartDate, policyDescription, policyName, dueDate)

INVENTORY (inventoryID, inventoryName)

INVOICE (invoiceID, paymentMode, status, State, Street, City, ZIP, orderID)

- *FOREIGN KEY* (orderID) References **ORDERS** (orderID)

ITEMS (itemID, itemName, itemDescription, brand, price, discountPrice, prodTypeID, vendorID)

- *FOREIGN KEY* (prodTypeID, vendorID) References **CAN_SUPPLY** (prodTypeID, vendorID)

ITEMS SUBCLASSES

SHOPPING_PRODUCTS (itemID, color, size, productCategory)

- *FOREIGN KEY* (itemID) References **ITEMS** (itemID)

PHARMACY_PRODUCTS (itemID, dosagePerDay, dosageTime, treatmentType)

- *FOREIGN KEY* (itemID) References **ITEMS** (itemID)

END OF ITEMS SUBCLASSES

LOCAL_SHELTER (shelterID, shelterName, contactName, contactEmail, contactNum, State, Street, City, ZIP)

LOGINS (username, password, employeeID)

- *FOREIGN KEY* (employeeID) References **EMPLOYEES** (employeeID)

ORDERS (orderID, orderDate, deliveryPrice, status, orderPrice, totalAmount, customerID)

- *FOREIGN KEY* (customerID) References **CUSTOMERS** (customerID)

PETS (petID, petName, petType, petBirthday, petAge, adoptionDate, petBreed, petWeight, customerID)

- *FOREIGN KEY* (customerID) References **CUSTOMERS** (customerID)

PRODUCT_TYPE (prodTypeID, prodTypeDesc)

TICKETS (ticketID, ticketDesc, ticketStatus, ticketPriority, ticketStart, ticketEnd, processingTime, department, customerID, employeeID)

- *FOREIGN KEY* (customerID) References **CUSTOMERS** (customerID)
- *FOREIGN KEY* (employeeID) References **CUSTOMER_SUPPORT_AGENTS** (employeeID)

TRAINING_COURSE (courseID, courseDesc, courseStartDate, courseDuration, employeeID)

- *FOREIGN KEY* (employeeID) References **DEVELOPERS** (employeeID)

VACCINE_TYPES (vacID, vacName)

VENDORS (vendorID, vendorName, contactName, contactEmail, contactNum, State, Street, City, ZIP)

VENDOR_TYPES(vtypeID, vtypeName)

VET_SERVICES (serviceID, serviceType, serviceLocation, serviceModality, serviceOrderDate, orderID)

- *FOREIGN KEY* (orderID) References **ORDERS** (orderID)

VET_SERVICES SUBCLASSES

DENTAL (serviceID, procedureType, prescription)

- *FOREIGN KEY* (serviceID) References **VET_SERVICES** (serviceID)

DENTAL_TOOLS_USED (serviceID, tool)

- *FOREIGN KEY* (serviceID) References **DENTAL** (serviceID)

GROOMING_PRODUCTS (serviceID, productUsed)

- *FOREIGN KEY* (serviceID) References **VET_SERVICES** (serviceID)

GROOMING_TYPE (serviceID, groomingType)

- *FOREIGN KEY* (serviceID) References **VET_SERVICES** (serviceID)

PARASITE_CONTROL (serviceID, parasiteType, medicine, dose)

- *FOREIGN KEY* (serviceID) References **VET_SERVICES** (serviceID)

VACCINATIONS (serviceID, vacDate, nextDueVacID, nextVacDate, vacID)

- *FOREIGN KEY* (serviceID) References **VET_SERVICES** (serviceID)
- *FOREIGN KEY* (vacID) References **VACCINE_TYPES** (vacID)

END OF VET_SERVICES SUBCLASSES

WEAK ENTITIES & RELATIONSHIPS

ASSOCIATES_CONNECTS_SHELTERS (shelterID, employeeID)

- *FOREIGN KEY* (employeeID) References **ADOPTION_ASSOCIATES** (employeeID)
- *FOREIGN KEY* (shelterID) References **LOCAL_SHELTER** (shelterID)

CAN_SUPPLY(prodTypeID, vendorID)

- *FOREIGN KEY* (vendorID) references to **VENDORS** (vendorID)
- *FOREIGN KEY* (prodTypeID) references to **PRODUCT_TYPE** (prodTypeID)

INSURANCE_PROVIDED (policyID, employeeID)

- *FOREIGN KEY* (employeeID) References **INSURANCE_AGENTS** (employeeID)
- *FOREIGN KEY* (policyID) References **INSURANCE** (policyID)

INVENTORY_DETAILS(itemID, inventoryID, itemQuantity, haveSupply)

- *FOREIGN KEY* (itemID) References **ITEMS** (itemID)
- *FOREIGN KEY* (inventoryID) References **INVENTORY** (inventoryID)

ITEMS_DETAILS (orderID, itemID, itemQuantity)

- *FOREIGN KEY* (orderID) References **ORDERS** (orderID)
- *FOREIGN KEY* (itemID) References **ITEMS** (itemID)

SERVICE_DETAILS (employeeID, serviceID, serviceStartTime, serviceEndTime, serviceDuration)

- *FOREIGN KEY* (orderID) references **ORDERS** (orderID)
- *FOREIGN KEY* (employeeID) references **VETERINARIANS** (employeeID)

VENDOR_CONTENTS (vendorID, vtypeID)

- *FOREIGN KEY* (vendorID) References to **VENDORS** (vendorID)
- *FOREIGN KEY* (vtypeID) References to **VENDOR_TYPES** (vtypeID)

Normalized Tables

TABLE NAME	PRIMARY & FOREIGN KEY
ADOPTION_ASSOCIATES	PK: employeeID FK: employeeID References EMPLOYEES (employeeID)
ADOPTION_ASSOCIATE_TRAININGS	PK: employeeID, trainingCertificateID FK: employeeID References ADOTPTION_ASSOCIATES
ASSOCIATES_CONNECTS_SHELTERS	PK: shelterID, employeeID FK: employeeID References ADOPTION_ASSOCIATES shelterID References LOCAL_SHELTER
CAN_SUPPLY	PK: prodTypeID, vendorID FK: vendorID references to VENDORS prodTypeID references to PRODUCT_TYPE
CUSTOMERS	PK: customerID
CUSTOMER_PHONES	PK: customerID, custPhone FK: customerID References CUSTOMERS
CUSTOMER_SUPPORT_AGENTS	PK: employeeID FK: employeeID References EMPLOYEES
CUST_SUPPORT_AGENTS_LANGUAGES	PK: employeeID, languageName FK: employeeID References CUSTOMER_SUPPORT
CUST_SUPPORT_AGENTS_STATES	PK: employeeID, stateCovered FK: employeeID References CUSTOMER_SUPPORT
DENTAL	PK: serviceID FK: (serviceID) References VET_SERVICES
DENTAL_TOOLS_USED	PK: serviceID FK: (serviceID) References DENTAL
DEVELOPERS	PK: employeeID FK: employeeID References EMPLOYEES
DEVELOPER_CODING	PK: employeeID, techLanguageName FK: employeeID References DEVELOPERS
EMPLOYEES	PK: employeeID

	FK: employeeID References to MANAGERS
EMPLOYEE_PHONES	PK: employeeID, employeePhone FK: employeeID References EMPLOYEES
GROOMING_TYPE	PK: serviceID FK: (serviceID) References VET_SERVICES
INSURANCE_AGENTS	PK:employeeID FK:employeeID) References EMPLOYEES
INSURANCE	PK: policyID
INSURANCE_PROVIDED	PK: policyID, employeeID FK: employeeID References INSURANCE_AGENTS policyID References INSURANCE
INVENTORY	PK: inventoryID
INVENTORY_DETAILS	PK: itemID, inventoryID FK: itemID References ITEMS inventoryID References INVENTORY
INVOICES	PK: invoiceID FK: orderID References ORDERS
ITEMS	PK: itemID FK: prodTypeID, vendorID References CAN_SUPPLY
ITEMS_DETAILS	PK: orderID, itemID FK: orderID References ORDERS itemID References ITEMS
LOCAL_SHELTERS	PK: shelterID
LOGINS	PK: username FK: employeeID References EMPLOYEES
MANAGERS	PK: employeeID FK: employeeID References to EMPLOYEES
ORDERS	PK: orderID FK: customerID References CUSTOMERS
PARASITE_CONTROL	PK: serviceID FK: (serviceID) References VET_SERVICES
PETS	PK: petID FK: customerID References CUSTOMERS
PHARMACY_PRODUCTS	PK: itemID FK: itemId References ITEMS
PRODUCT_TYPE	PK: prodTypeID
SERVICE_DETAILS	PK: employeeID, serviceID, serviceStartTime FK: orderID references ORDERS employeeID references VETERINARIANS
SHOPPING_PRODUCTS	PK: itemID FK: itemId References ITEMS

TICKETS	PK: ticketID FK: customerID References CUSTOMERS employeeID References CUSTOMER_SUPPORT_AGENTS
TRAINING_COURSE	PK: courseID FK: employeeID References DEVELOPERS
VACCINATIONS	PK: serviceID FK: serviceID References VET_SERVICES vacID References VACCINE_TYPES
VACCINE_TYPES	PK: vtypeID
VENDORS	PK: vendorID
VENDOR_CONTENTS	PK: vendorID, vtypeID FK: vendorID References to VENDORS vtypeID) References to VENDOR_TYPES
VET_SERVICES	PK: serviceID FK: orderID References ORDERS
VETERINARIANS	PK: employeeID FK: employeeID References EMPLOYEES
VETERINARIANS_CERTIFICATES	PK: employeeID, vetCertificateID FK: employeeID References VETERINARIANS

Chapter 4: SQL Queries

1. Top N Items by sale

```
SELECT I.ITEMID, BRAND, ITEMDESC, PRICE as "PRICE_PER_ITEM",
    SUM(ITEMQUANTITY) AS ITEM_COUNT
FROM ITEMS I
JOIN ITEM_DETAILS ID ON I.ITEMID = ID.ITEMID
GROUP BY I.ITEMID, ITEMDESC, PRICE, BRAND
ORDER BY ITEM_COUNT DESC
FETCH FIRST 5 ROWS WITH TIES;
```

Output:

Query Result

SQL | All Rows Fetched: 5 in 0.087 seconds

	ITEMID	BRAND	ITEMDESC	PRICE_PER_ITEM	ITEM_COUNT
1	I109	Barefoot	RoundBed	25	26
2	I104	Tuffy	ChewToy	10	24
3	I101	Nordog	DogTags	18	22
4	I112	Tuffy	RopeToy	11	22
5	I102	Nordog	DogHoodie	11	21

Explanation : *This query displays all the top N items sold by item quantity using the items ordered data in the table ITEMS_DETAILS.*

2. Vaccination Tracker

```
WITH temp_vac AS (
            SELECT c.CUSTOMERID AS Customer_ID, CUSTUSERNAME AS
Customer_Name, CUSTEMAIL AS Customer_Email,
            CUSTPHONE AS Customer_Phone, PETID AS Pet_ID, PETNAME AS
Pet_Name, vs.SERVICEID AS Service_ID,
            NEXTDUEVACID AS Next_Vaccine_ID, VACNAME AS Vaccine_Name,
VACDATE AS Previous_Vaccine_Date, NEXTVACDATEDUE AS
Next_Vacccine_Due_Date
            FROM CUSTOMERS c
            JOIN CUSTOMER_PHONES cp ON c.CUSTOMERID = cp.CUSTOMERID
            JOIN PETS p ON c.CUSTOMERID = p.CUSTOMERID
            JOIN ORDERS o ON c.CUSTOMERID = o.CUSTOMERID
            JOIN VET_SERVICES vs ON o.ORDERID = vs.ORDERID
            JOIN VACCINATIONS v ON vs.SERVICEID = v.SERVICEID
            JOIN VACCINATION_TYPES vt ON v.NEXTDUEVACID = vt.VACID
            WHERE NEXTVACDATEDUE >= sysdate AND (NEXTVACDATEDUE <=
TRUNC(SYSDATE+14))
            )
            SELECT DISTINCT Customer_ID "Customer ID",
                            Customer_Name "Name",
                            Customer_Email "Email",
                            Customer_Phone "Phone",
                            Pet_ID "Pet ID",
                            Pet_Name "Pet Name",
                            Next_Vaccine_ID "Vaccination ID",
                            Next_Vacccine_Due_Date "Next Due Date"
            FROM temp_vac tv
            JOIN vaccinations v1 ON tv.Next_Vaccine_ID =
v1.NEXTDUEVACID
            WHERE Previous_Vaccine_Date IN (
                SELECT vacdate
                FROM vaccinations v2
                JOIN vet_services vs2 ON v2.SERVICEID = vs2.SERVICEID
                JOIN orders o2 ON vs2.ORDERID = o2.ORDERID
                WHERE NEXTVACDATEDUE IS NOT NULL
            )
```

Output:

	Customer ID	Name	Email	Phone	Pet ID	Pet Name	Vaccination ID	Next Due Date
1	C10025	MadisonKim	madisonkim22@gmail.com	3794378292	P10021	Cashew	136	17-DEC-2022
2	C10011	unnati_pal@131	unnatipal131@gmail.com	9208250505	P10022	Kulfi	543	21-DEC-2022

Explanation: *Vaccination tracker displays the customer and pets' data for the range of date selected by the admin. If the admin selects 14 days range, the query shows the pets that are due to get a booster vaccine in that date range.*

3. Customer Analytics By Brand

```
WITH cus AS (SELECT C.CUSTOMERID,FNAME ||' '|| LNAME AS NAME,
        brand,count(brand)
        FROM CUSTOMERS C
        JOIN ORDERS O ON C.CUSTOMERID = O.CUSTOMERID
        JOIN ITEM_DETAILS ID ON ID.ORDERID = O.ORDERID
        JOIN ITEMS I ON I.ITEMID = ID.ITEMID
        JOIN PRODUCT_TYPES PT ON PT.PRODTYPEID = I.PRODTYPEID
        WHERE ORDERDATE > (SYSDATE)-40
        group by C.CUSTOMERID,FNAME ||' '|| LNAME,brand
        ),
        pv as (SELECT * FROM cus
        PIVOT(
           count(customerid)
          FOR Brand IN ('Buddy Wash' AS BuddyWash ,'Barefoot' AS
Barefoot,'Nordog' AS Nordog,
           'Wellness' AS Wellness,'Penguin' AS Penguin,'Laboni' AS
Laboni,'Benebone' AS Benebone,'Tuffy' AS Tuffy,
           'Pedigree' AS Pedigree,'Casper' AS Casper,'PetAg' AS
PetAg,'Nylabone' AS Nylabone)
    )
        ),
        r2 as (select FNAME ||' '|| LNAME AS NAME,count(brand) as ttt
FROM CUSTOMERS C
        JOIN ORDERS O ON C.CUSTOMERID = O.CUSTOMERID
        JOIN ITEM_DETAILS ID ON ID.ORDERID = O.ORDERID
        JOIN ITEMS I ON I.ITEMID = ID.ITEMID
        JOIN PRODUCT_TYPES PT ON PT.PRODTYPEID = I.PRODTYPEID
        WHERE ORDERDATE > (SYSDATE)-40
        group by FNAME ||' '|| LNAME
        )
```

```
        select p.name,BuddyWash,Barefoot,Nordog,Wellness,Penguin,
Laboni,

        Benebone,Tuffy,Pedigree, Casper, PetAg, Nylabone,ttt from pv p
join r2 r on p.name = r.name

        order by ttt desc fetch first 5 row only
```

Query Result

All Rows Fetched: 5 in 0.063 seconds

	NAME	BUDDYWASH	BAREFOOT	NORDOG	WELLNESS	PENGUIN	LABONI	BENEBONE	TUFFY	PEDIGREE	CASPER	PETAG	NYLABONE	TTT
1	Angelica Putman	0	1	1	0	1	0	0	1	0	1	0	1	6
2	James Johnson	1	0	1	0	0	0	0	0	1	0	0	0	3
3	Wilma Anderson	0	0	0	0	1	0	0	1	1	0	0	0	3
4	Charles Lopez	0	0	1	0	0	0	0	0	0	0	0	0	2
5	Madison Kim	0	0	1	0	0	0	0	1	0	0	0	0	2

Explanation: *Customer Analytics by brand query gives insight into top customers buying which brand . It will also calculate the total of the brand to understand which customer buys the most items.*

4. Total Price of Orders

```
WITH pr AS (

    SELECT

        id.itemquantity, o.orderid, fname, lname, c.state,c.city,
c.street,price, deliveryprice, count(c.customerid), c.customerid

    FROM

        invoice i

        JOIN orders o ON o.orderid = i.orderid

        JOIN customers c ON c.customerid = o.customerid

        JOIN item_details id ON id.orderid = o.orderid

        JOIN items i ON i.itemid = id.itemid

    GROUP BY fname, lname, c.state, c.city, c.street, price,
deliveryprice, c.customerid, id.itemquantity,o.orderid

),

que AS(

SELECT customerid, orderid,

    SUM(CASE

        WHEN itemquantity > 1 THEN

            ( ( price * itemquantity ))
```

```
        ELSE
            price
    END     )  TotalPrice
FROM
    pr
GROUP BY customerid,orderid
)
SELECT q.customerid, q.orderid, q.TotalPrice + o.deliveryprice
FROM que q
INNER JOIN customers c
    ON c. customerid = q.customerid
INNER JOIN orders o
    ON o.orderid = q.orderid;
```

Query Result

SQL | All Rows Fetched: 17 in 0.124 seconds

	CUSTOMERID	ORDERID	Q.TOTALPRICE+O.DELIVERYPRICE
1	C10001	O10028	163
2	C10005	O10029	103
3	C10006	O10030	210
4	C10007	O10031	740
5	C10008	O10032	494
6	C10011	O10033	30
7	C10014	O10034	1056
8	C10019	O10035	60
9	C10025	O10036	582
10	C10003	O10037	30
11	C10006	O10038	455
12	C10009	O10039	36
13	C10013	O10040	110
14	C10017	O10041	34
15	[illegible]	[illegible]	[illegible]

Explanation: *Total price of Orders query calculate price of the order by taking the item price quantity of that item into consideration and adding a delivery charge on a particular order.*

5. Employee Count Under a Manager

```
WITH q AS(
```

```
SELECT e.managerid,
       count(e.employeeid) AS No_of_Employees
 FROM EMPLOYEES e
 WHERE managerid IS NOT NULL
  GROUP BY managerid
  ORDER BY count(e.employeeid)
  )
  SELECT   e.employeeid,
           e.DEPARTMENT,
           e.DESIGNATION,
           e.fname||' '||mname||' '||lname AS Name,
           e.employeeemail AS Email,
           c.EMPLOYEEPHONE AS Contact,
           q.No_of_Employees AS "No of Employees"
FROM EMPLOYEES e
JOIN EMPLOYEE_PHONES c
    ON c.EMPLOYEEID = e.EMPLOYEEID
JOIN q q
    ON q.managerid = c.employeeid
WHERE designation = 'MANAGER'
```

Query Result

All Rows Fetched: 9 in 0.055 seconds

	EMPLOYEEID	DEPARTMENT	DESIGNATION	NAME	EMAIL	CONTACT	No of Employees
1	E10088	MARKETING	MANAGER	Mohammad Jafar	mjafar@chicago.edu	6309391312	1
2	E10015	IT	MANAGER	Nick John	nickjohn@chewy.com	3813318173	3
3	E10020	SALES	MANAGER	Jonathan Alford	jonathanalford@chewy.com	2352467775	3
4	E10089	MARKETING	MANAGER	Mohammad Jafar	mj@chicago.edu	5346437433	3
5	E10009	SALES	MANAGER	Alex Camp	alexcamp@chewy.com	6468681693	3
6	E10019	MARKETING	MANAGER	Hill Chen	hillchen@chewy.com	3234345435	4
7	E10016	SALES	MANAGER	Riya Chess	riyachess@chewy.com	4134135556	4
8	E10079	SALES	MANAGER	Varun K	varun@chewy.com	6078909100	5
9	E10006	HR	MANAGER	Mehul Bendal	mehulbendal@chewy.com	5202215699	5

Explanation: *Fetches every manager's contact information along with the number of people that that manager is supervising.*

6. Inventory Details

```
SELECT itemDesc, INVENTORYNAME, COUNT(I.itemDesc)
FROM
    ITEMS I
    JOIN INVENTORY_DETAILS ID ON I.ITEMID = ID.ITEMID
    JOIN INVENTORY INV ON ID.INVENTORYID = INV.INVENTORYID
GROUP BY
    itemDesc, INVENTORYNAME
HAVING COUNT(I.itemDesc) > (
    SELECT
        FLOOR(AVG(total_items))
    FROM
        (
            SELECT
                COUNT(I.itemDesc) AS "TOTAL_ITEMS"
            FROM
                ITEMS I
                JOIN INVENTORY_DETAILS ID ON I.ITEMID = ID.ITEMID
                JOIN INVENTORY INV ON ID.INVENTORYID = INV.INVENTORYID
            GROUP BY
                itemDesc, INVENTORYNAME
        )
)
ORDER BY I.itemDesc;
```

Query Result

All Rows Fetched: 2 in 0.058 seconds

	ITEMDESC	INVENTORYNAME	COUNT(I.ITEMDESC)
1	CannedFood	Phoenix Fulfillment Center	2
2	ChewToy	Tucson Fulfillment Center	3

Explanation: *Top Fulfillment Centers that have more supply of item types than the average of all item count.*

7. Most expensive product under each category

```
SELECT
    a.PRODTYPEID AS "Product Type ID",
    a.PRODTYPE AS "Product Type",
    max_list_price AS "Maximum Priced Item"
FROM
    PRODUCT_TYPES a,
    (
        SELECT
            PRODTYPEID,
            MAX( price ) max_list_price
        FROM
            items
        GROUP BY
            PRODTYPEID
    ) b
WHERE
    a.PRODTYPEID = b.PRODTYPEID
ORDER BY
    PRODTYPE;
```

Query Result

All Rows Fetched: 6 in 0.044 seconds

	Product Type ID	Product Type	Maximum Priced Item
1	PT101	Accessories	18
2	PT103	Beds	25
3	PT102	Clothing	12
4	PT100	Food	27
5	PT105	Grooming	49
6	PT104	Toys	15

Explanation: *Fetches the cost of item which is the most expensive product under that category*

8. Employee count within each department

```
with q as ( SELECT COUNT(*) AS num_of_emp,
                   department
                   FROM EMPLOYEES emp
                   GROUP BY department
)
SELECT e.department,
       emp.num_of_emp AS "Number of Employees",
       LISTAGG(e.fname ||' '|| e.mname||' ' || e.lname, ',')
WITHIN GROUP (ORDER BY e.fname) AS "Employee Names"
FROM   employees e
JOIN q emp
    ON emp.department = e.department
GROUP BY e.department, emp.num_of_emp;
```

Query Result

All Rows Fetched: 5 in 0.037 seconds

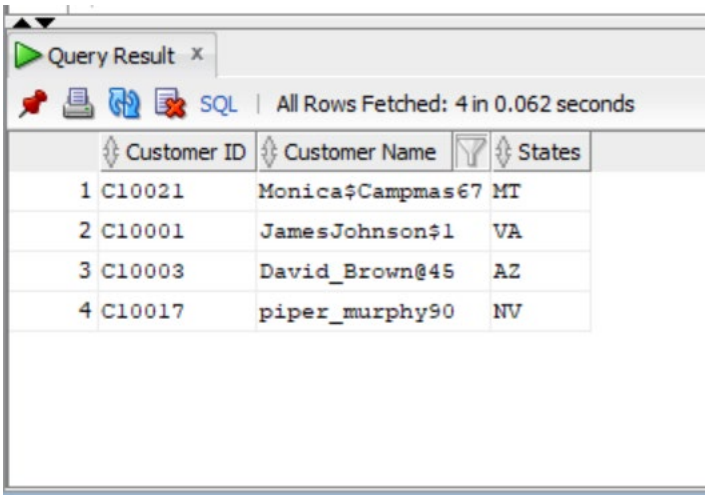
	DEPARTMENT	Number of Employees	Employee Names
1	HR	14	Abhishek Nanoti,Anna Bella,Bhargavi ,Ellie Hilton,Ellie Hilton,Joe Rogan,Marsha Jonas,Mehul Bendal
2	IT	11	Bailey Naray,Den Jack,Franklin Benjamin,Jason Mason,Jass Kell,Kelly Jacobs,Maroon K,Max ,Nick Joh
3	ADMIN	8	Bhargavi Murlidhara,Jaimin Fojdar,Max Goswitz,Medha Singh,Medha Singh,Tanishka Shorey,Unnati Paland
4	SALES	23	Ak ,Alex Camp,Danise James,Danise James,Danise James,Danise James,Fred Potter,Ira Sharma,John ,Jo
5	MARKETING	9	Daisy Jones,Dalton Thomson,Hill Chen,Mohammad Jafar,Mohammad Jafar,Patrick Stout,Priyanka Verma,Tre

Explanation: *Fetches number of employees in each department with their full names.*

9. Frequent Customer

```
WITH generalOrders AS (
SELECT C.CUSTOMERID, COUNT(DISTINCT O.OrderID) AS overallOrders
FROM CUSTOMERS C JOIN ORDERS O ON C.CUSTOMERID = O.CUSTOMERID
GROUP BY C.CUSTOMERID
)
SELECT C.CUSTOMERID AS "Customer ID", C.CUSTUSERNAME AS "Customer
Name",
C.STATE AS "States"
FROM CUSTOMERS C JOIN ORDERS O ON C.CUSTOMERID = O.CUSTOMERID
JOIN generalOrders ON C.CUSTOMERID = generalOrders.CUSTOMERID
AND O.CUSTOMERID IN (
SELECT O.CUSTOMERID
FROM ORDERS O
WHERE EXTRACT(YEAR FROM OrderDate) = '2022'
GROUP BY O.CUSTOMERID
HAVING COUNT(O.CUSTOMERID) >= 2
)
AND overallOrders > (
SELECT MEDIAN(overallOrders)
FROM generalOrders
)
```

```
GROUP BY C.CUSTOMERID, C.CUSTUSERNAME, C.STATE
;
```



Query Result

All Rows Fetched: 4 in 0.062 seconds

	Customer ID	Customer Name	States
1	C10021	Monica$Campmas67	MT
2	C10001	JamesJohnson$1	VA
3	C10003	David_Brown@45	AZ
4	C10017	piper_murphy90	NV

Explanation: *Frequent customers by states for the year 2022 and who have made more orders than the median of overall orders.*

10. Most profitable month

```
with p as (
SELECT  EXTRACT(YEAR FROM orderdate) AS year,
        EXTRACT(month FROM orderdate) AS month,
        COUNT(orderid) AS number_of_orders
FROM orders
GROUP BY EXTRACT(YEAR FROM orderdate), EXTRACT(month FROM orderdate)
ORDER BY EXTRACT(YEAR FROM orderdate), EXTRACT(month FROM orderdate)
),
q as(
select year,month,number_of_orders,
rank() over(order by number_of_orders desc) AS most_profitable_month
from p
)
select *
from q
```

```
fetch first 5 rows only

;
```

Query Result

All Rows Fetched: 5 in 0.032 seconds

	YEAR	MONTH	NUMBER_OF_ORDERS	MOST_PROFITABLE_MONTH
1	2022	11	21	1
2	2021	11	5	2
3	2021	12	4	3
4	2021	6	3	4
5	2022	7	2	5

Explanation: *Most profitable month query extracts month and year from the order date and calculates number of products sold in that month and year and then rank them. The month and year with the highest number of products sold is the most profitable month.*

11. Item price which are less than average price

```
SELECT
    itemid,
    itemdesc,
    price
FROM
   items
WHERE
    price (
        SELECT
            AVG(price)
        FROM
            items
    )
    order by price;
```

Query Result

All Rows Fetched: 17 in 0.055 seconds

	ITEMID	ITEMDESC	PRICE
1	I118	ChewToy	10
2	I114	ChewToy	10
3	I104	ChewToy	10
4	I110	ChewToy	10
5	I102	DogHoodie	11
6	I120	RopeToy	11
7	I112	RopeToy	11
8	I123	RopeToy	12

12. Subtotals of Items for each state

```
SELECT STATE, ITEMDESC, SUM(PRICE) AS "Item Price"
FROM ITEMS I
JOIN ITEM_DETAILS ID ON ID.ITEMID = I.ITEMID
JOIN ORDERS O ON O.ORDERID = ID.ORDERID
JOIN CUSTOMERS C ON C.CUSTOMERID = O.CUSTOMERID
GROUP BY ROLLUP(STATE, ITEMDESC)
;
```

Query Result

All Rows Fetched: 37 in 0.038 seconds

	STATE	ITEMDESC	Item Price
1	AR	CannedFood	15
2	AR	(null)	15
3	AZ	ChewToy	10
4	AZ	DogTags	18
5	AZ	DogHoodie	11
6	AZ	(null)	39
7	CA	RoundBed	25
8	CA	Conditioner	49

Explanation: *Total revenue by each state and item type and calculating the subtotal of it for our Marketing team to perform analysis.*

13. Subtotals of Items for each state

```
WITH cus AS (
    SELECT
        c.customerid,
        fname
        || ' '
        || lname AS name,
        prodtype
    FROM
             customers c
        JOIN orders        o ON c.customerid = o.customerid
        JOIN item_details  id ON id.orderid = o.orderid
        JOIN items         i ON i.itemid = id.itemid
        JOIN product_types pt ON pt.prodtypeid = i.prodtypeid
    WHERE
        orderdate > ( sysdate ) - 40
), pv AS (
    SELECT
        *
    FROM
        cus PIVOT (
            COUNT(customerid)
            FOR prodtype
            IN ( 'Food' AS food, 'Beds' AS beds, 'Accessories' AS
accessories, 'Clothing' AS clothing, 'Toys' AS toys, 'Grooming' AS
grooming
            )
        )
), r2 AS (
    SELECT
```

```
        fname || ' ' || lname        AS name,
        COUNT(prodtype) AS total
    FROM
             customers c
        JOIN orders        o ON c.customerid = o.customerid
        JOIN item_details  id ON id.orderid = o.orderid
        JOIN items         i ON i.itemid = id.itemid
        JOIN product_types pt ON pt.prodtypeid = i.prodtypeid
    WHERE
        orderdate > ( sysdate ) - 40
    GROUP BY
        fname
        || ' '
        || lname
)
SELECT p.name, food, beds, accessories, clothing, toys, grooming,
total
FROM
         pv p
    JOIN r2 r ON p.name = r.name
ORDER BY
    total DESC
FETCH FIRST 5 ROW ONLY;
```

	NAME	FOOD	BEDS	ACCESSORIES	CLOTHING	TOYS	GROOMING	TOTAL
1	Angelica Putman	1	2	0	1	2	0	6
2	James Johnson	1	0	1	0	0	1	3
3	Wilma Anderson	2	0	0	0	1	0	3
4	Charles Lopez	0	0	1	1	0	0	2
5	Madison Kim	0	0	1	0	1	0	2

Explanation: *Customer Analytics by product type query gives insight into top customers buying which product type. It will also calculate the total of the product type to understand which customer buys the most items.*

Chapter 5: Triggers and Procedures

1. Discount Price for the requested Brand and Product Type

```
create or replace PROCEDURE discountprice (
    discount    NUMBER,
    in_brand    items.brand%TYPE,
    in_prodtype product_types.prodtype%TYPE
) AS

    CURSOR c1 IS
    SELECT
        itemid,
        price,
        brand,
        prodtype
```

```
    FROM
             items i
        JOIN product_types pt ON i.prodtypeid = pt.prodtypeid
    WHERE
            brand = in_brand
        AND prodtype = in_prodtype;

    counttype         NUMBER;
    temp_itemid       items.itemid%TYPE;
    temp_brand        items.brand%TYPE;
    temp_prodtype     product_types.prodtype%TYPE;
    v_discountedprice items.discountedprice%TYPE;
    temp_price        items.price%TYPE;
BEGIN
    SELECT
        COUNT(pt.prodtypeid)
    INTO counttype
    FROM
             items i
        JOIN product_types pt ON i.prodtypeid = pt.prodtypeid
    WHERE
            brand = in_brand
        AND prodtype = in_prodtype;

    IF counttype > 0 THEN
        FOR rec IN c1 LOOP
            temp_itemid := rec.itemid;
            temp_brand := rec.brand;
            temp_prodtype := rec.prodtype;
            temp_price := rec.price;
            v_discountedprice := ( rec.price - ( rec.price * discount
/ 100 ) );
```

```
            UPDATE items
            SET
                discountedprice = v_discountedprice
            WHERE
                rec.itemid = itemid;

        END LOOP;

    ELSE
        raise_application_error(-20010, 'No Data Found');
    END IF;

END;
```

Before Execution

	ITEMID	ITEMDESC	PRICE	DISCOUNTEDPRICE	BRAND	PRODTYPEID
1	I100	DryFood	27	(null)	Pedigree	PT100
2	I101	DogTags	18	(null)	Nordog	PT101

After Execution

	ITEMID	ITEMDESC	PRICE	DISCOUNTEDPRICE	BRAND	PRODTYPEID
1	I100	DryFood	27	13.5	Pedigree	PT100
2	I101	DogTags	18	(null)	Nordog	PT101

Explanation: *Discount price procedure takes three user input*

1. *% of discount*
2. *Brand*
3. *Product type*

After taking this input it will calculate discount price on those products and update discount price.

2. Remove Discount Price for the requested Brand and Product Type

```
create or replace PROCEDURE removediscount (
    in_brand    items.brand%TYPE,
    in_prodtype product_types.prodtype%TYPE
) AS

    CURSOR c2 IS
    SELECT
        itemid,
        price,
        brand,
        prodtype
    FROM
             items i
        JOIN product_types pt ON i.prodtypeid = pt.prodtypeid
    WHERE
            brand = in_brand
        AND prodtype = in_prodtype;

    v_itemid items.itemid%TYPE;
    v_brand  items.brand%TYPE;
    v_type   product_types.prodtype%TYPE;
    v_price  items.price%TYPE;
BEGIN
    FOR rec IN c2 LOOP
        v_itemid := rec.itemid;
        v_brand := rec.brand;
        v_type := rec.prodtype;
        v_price := rec.price;
```

```
        UPDATE items
        SET
            discountedprice = 0
        WHERE
            itemid = v_itemid;

    END LOOP;
END;
```

Before Execution

	ITEMID	ITEMDESC	PRICE	DISCOUNTEDPRICE	BRAND	PRODTYPEID
1	I100	DryFood	27	13.5	Pedigree	PT100
2	I101	DogTags	18	(null)	Nordog	PT101

After Execution

	ITEMID	ITEMDESC	PRICE	DISCOUNTEDPRICE	BRAND	PRODTYPEID
1	I100	DryFood	27	0	Pedigree	PT100
2	I101	DogTags	18	(null)	Nordog	PT101

Explanation: *Remove Discount price procedure takes two user input*

1. *Brand*
2. *Product type*

After taking this input it will update discount price as '0' on those products.

3. Update Salary of the employees

```
create or replace PROCEDURE
    employeesalary (INCREMENT employees.salary%type, LIMIT
employees.salary%type) AS
IncrementedSal employees.salary%type := 0;
  CURSOR c1 IS
    SELECT employeeid, salary
  FROM employees FOR UPDATE OF salary;
  BEGIN
```

```
    FOR c1_rec IN c1 LOOP

IncrementedSal := c1_rec.salary + INCREMENT;

dbms_output.put_line('Proposed Salary for employee ' ||
c1_rec.employeeid || ' is ' || to_char(IncrementedSal) );

IF (IncrementedSal > LIMIT) THEN

dbms_output.put_line('Salary for employee ' || c1_rec.employeeid || '
currently ' || to_char(c1_rec.salary) || ' increment will put it over
limit of ' || to_char(LIMIT) );

ELSE

UPDATE employees SET salary = IncrementedSal

  WHERE CURRENT OF c1;

            dbms_output.put_line('   - Salary update successful for
employee ' || c1_rec.employeeid);

END IF;

END LOOP;

  END employeesalary;
```

Before Execution

	DESIGNATION	DEPARTMENT	MANAGERID	SSN	DATEOFHIRE	STATE	STREET	CITY	ZIP	SALARY
1	ASSOCIATE	SALES	E10001	(null)	26-NOV-2022	CA	1725N PARK AVE	SF	87654	50000
2	CEO	ADMIN	E10009	(null)	26-NOV-2022	AZ	1725N PARK AVE	Tucson	85719	120000
3	LEAD	MARKETING	E10044	(null)	26-NOV-2022	AZ	1450W PARK AVE	Tucson	89076	80000
4 com	ASSOCIATE	SALES	E10005	(null)	26-NOV-2022	AZ	1234N PARK AVE	Flagstaff	67540	50000
5 .com	ASSOCIATE	SALES	E10005	(null)	26-NOV-2022	AZ	1234N PARK AVE	Flagstaff	67540	50000

After Execution

	DESIGNATION	DEPARTMENT	MANAGERID	SSN	DATEOFHIRE	STATE	STREET	CITY	ZIP	SALARY
1	ASSOCIATE	SALES	E10001	(null)	26-NOV-2022	CA	1725N PARK AVE	SF	87654	51000
2	CEO	ADMIN	E10009	(null)	26-NOV-2022	AZ	1725N PARK AVE	Tucson	85719	121000
3	LEAD	MARKETING	E10044	(null)	26-NOV-2022	AZ	1450W PARK AVE	Tucson	89076	81000
4 com	ASSOCIATE	SALES	E10005	(null)	26-NOV-2022	AZ	1234N PARK AVE	Flagstaff	67540	51000
5 .com	ASSOCIATE	SALES	E10005	(null)	26-NOV-2022	AZ	1234N PARK AVE	Flagstaff	67540	51000
6 m	ASSOCIATE	SALES	E10009	(null)	04-DEC-2022	CA	1780S ELM STREET	SF	85719	51000

Explanation: *Update Employee Salary procedure takes two user input*

1. *Increment*
2. *Limit*

After taking this input it will increment salary of the employees whose salary is below the limit mentioned.

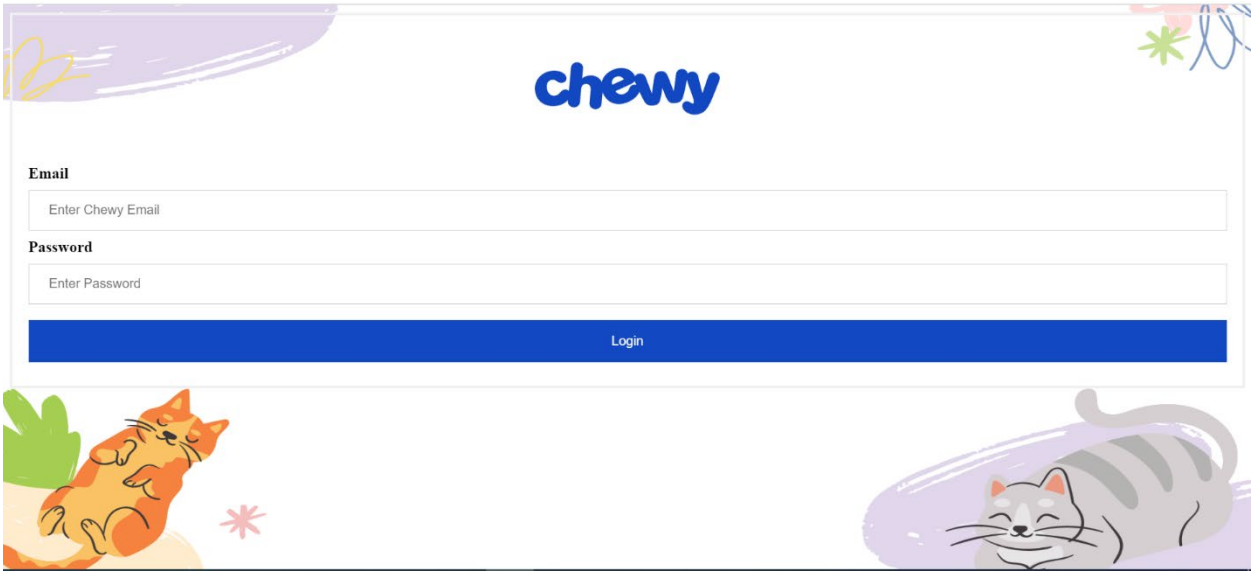
Chapter 6: User Interface

URL: 54.184.128.63/Chewy/login.html

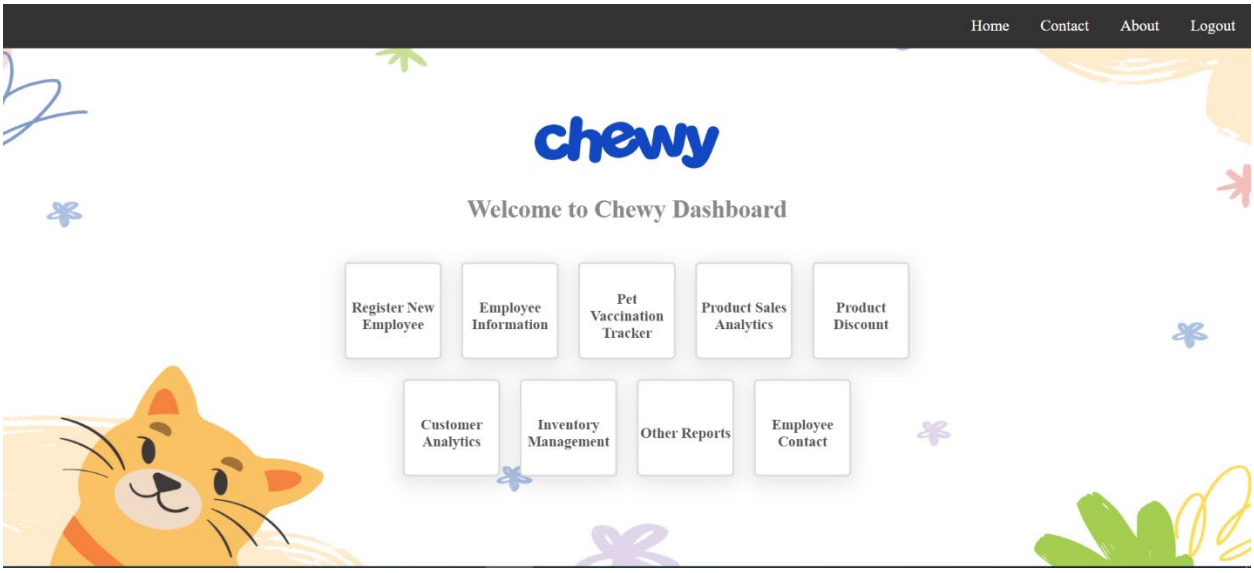
Login page:

Username: drcurrim@chewy.com

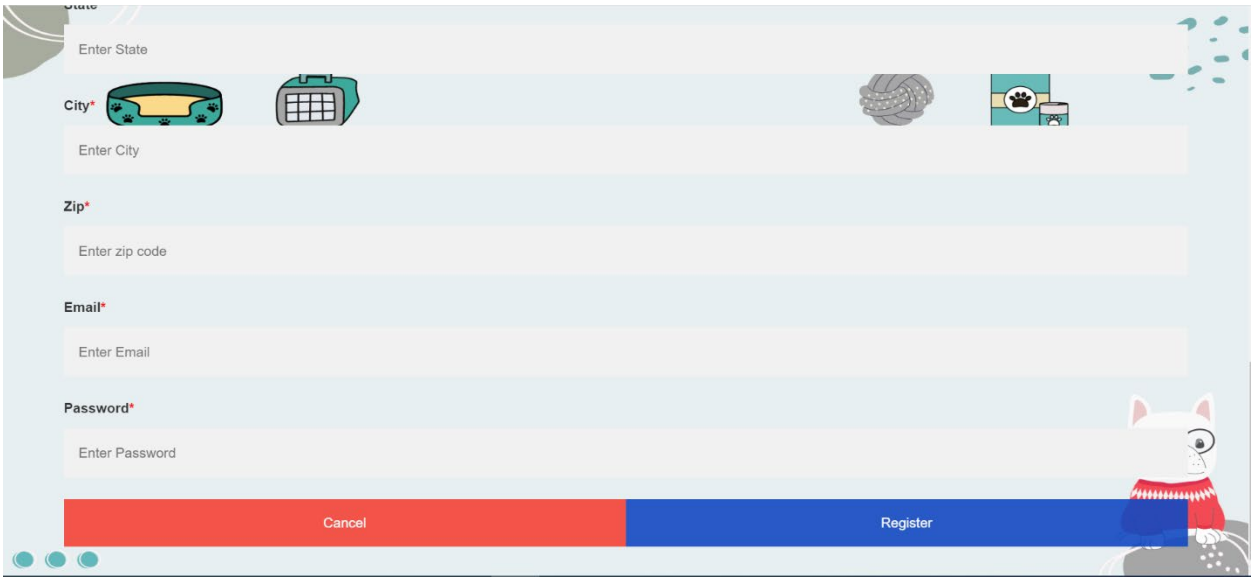
Password: Admin@531



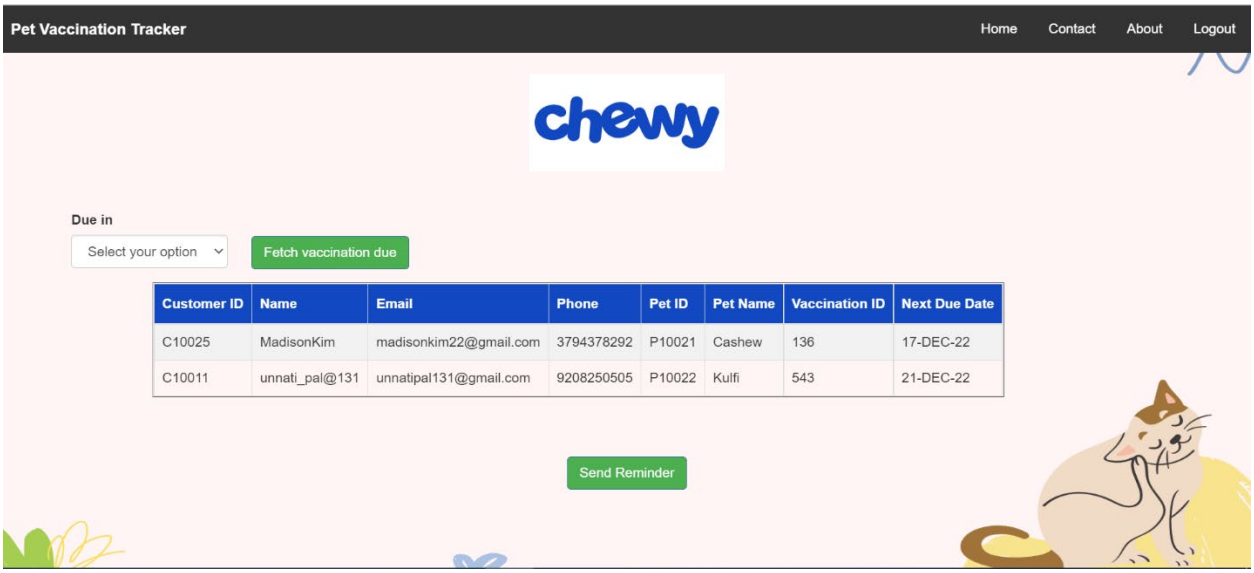
Dashboard: After sign-in the CHEWY dashboard opens which has all the built-in applications



Sign up page: Register New Employees application helps us to on-board new employees to CHEWY

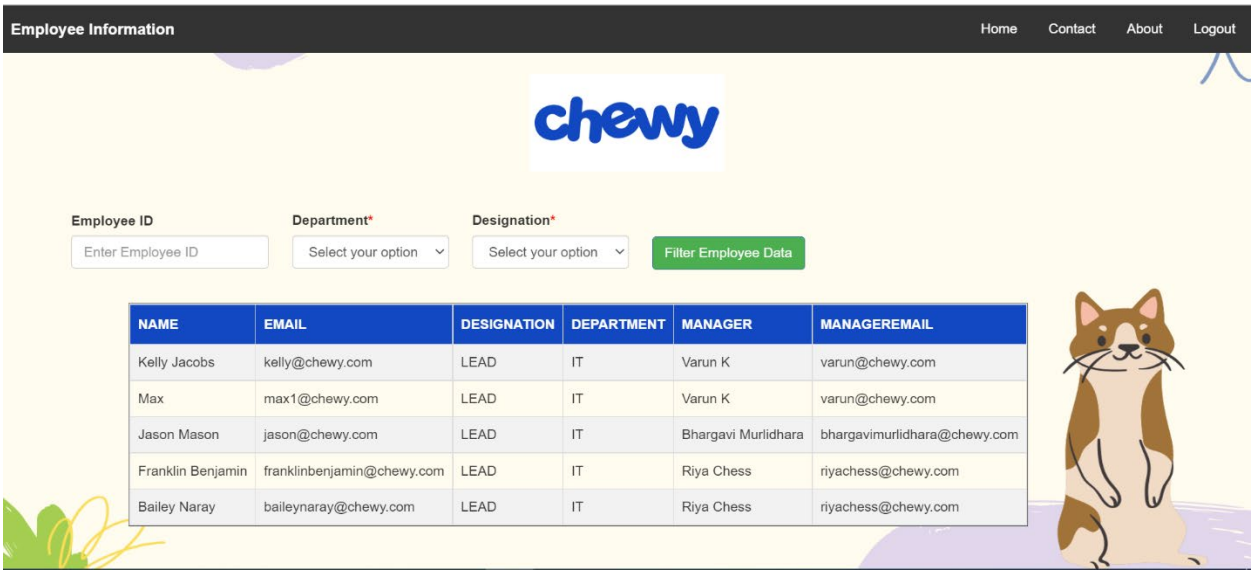


Vaccination Tracker: This application fetches the customer and pet details who are due to get vaccinated in next n days



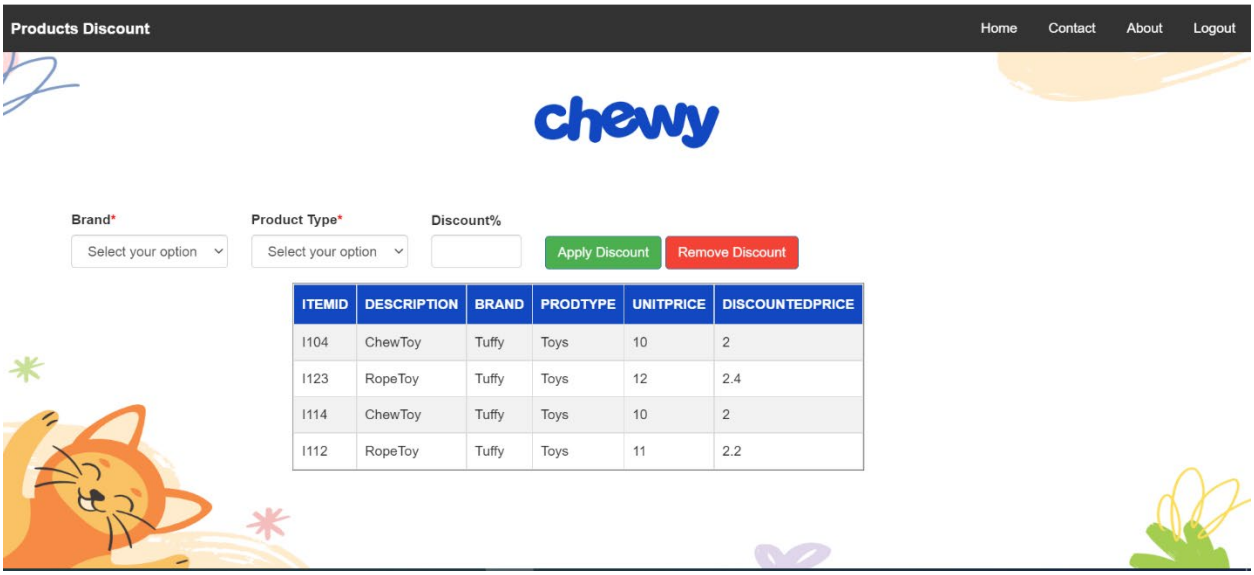
Customer ID	Name	Email	Phone	Pet ID	Pet Name	Vaccination ID	Next Due Date
C10025	MadisonKim	madisonkim22@gmail.com	3794378292	P10021	Cashew	136	17-DEC-22
C10011	unnati_pal@131	unnatipal131@gmail.com	9208250505	P10022	Kulfi	543	21-DEC-22

Employee Information: Fetches the employee and their manager details



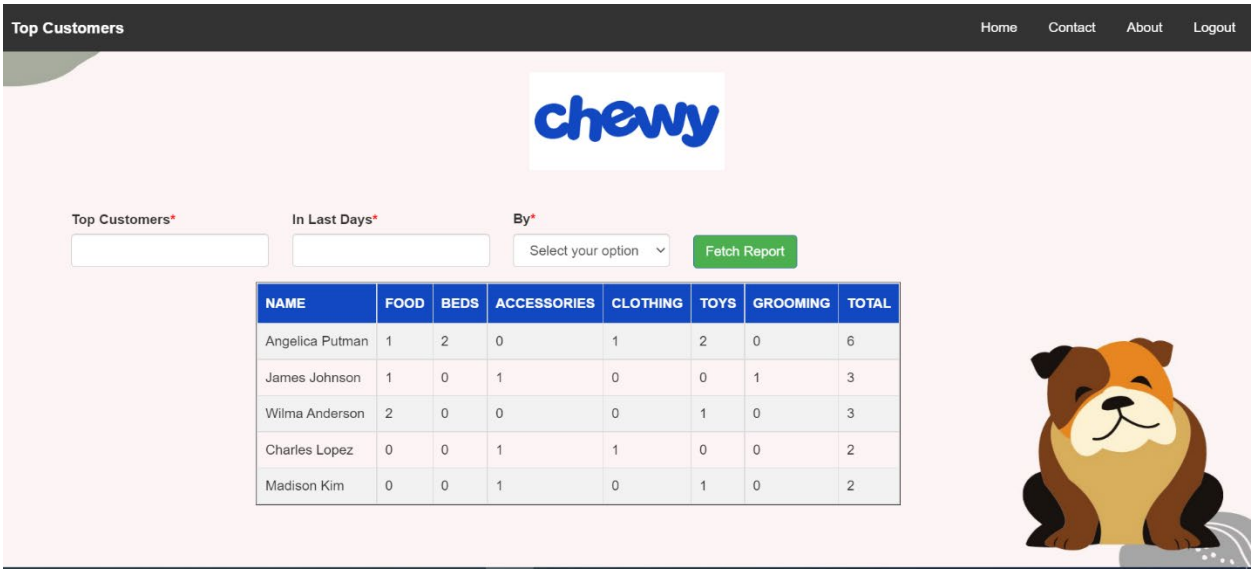
NAME	EMAIL	DESIGNATION	DEPARTMENT	MANAGER	MANAGEREMAIL
Kelly Jacobs	kelly@chewy.com	LEAD	IT	Varun K	varun@chewy.com
Max	max1@chewy.com	LEAD	IT	Varun K	varun@chewy.com
Jason Mason	jason@chewy.com	LEAD	IT	Bhargavi Murlidhara	bhargavimurlidhara@chewy.com
Franklin Benjamin	franklinbenjamin@chewy.com	LEAD	IT	Riya Chess	riyachess@chewy.com
Bailey Naray	baileynaray@chewy.com	LEAD	IT	Riya Chess	riyachess@chewy.com

Apply discount: Helps user to apply and remove discounts based on product category and brand



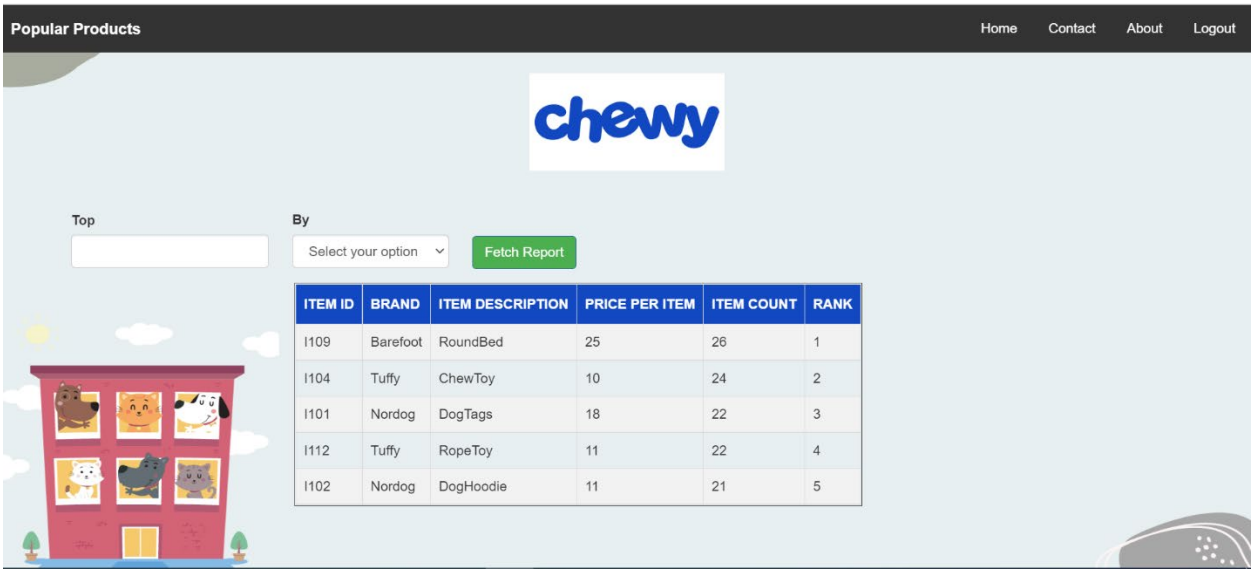
ITEMID	DESCRIPTION	BRAND	PRODTYPE	UNITPRICE	DISCOUNTEDPRICE
I104	ChewToy	Tuffy	Toys	10	2
I123	RopeToy	Tuffy	Toys	12	2.4
I114	ChewToy	Tuffy	Toys	10	2
I112	RopeToy	Tuffy	Toys	11	2.2

Customer analytics: Fetches top n customer who shop frequently from CHEWY in last n days and gives us a category and brand wise distribution



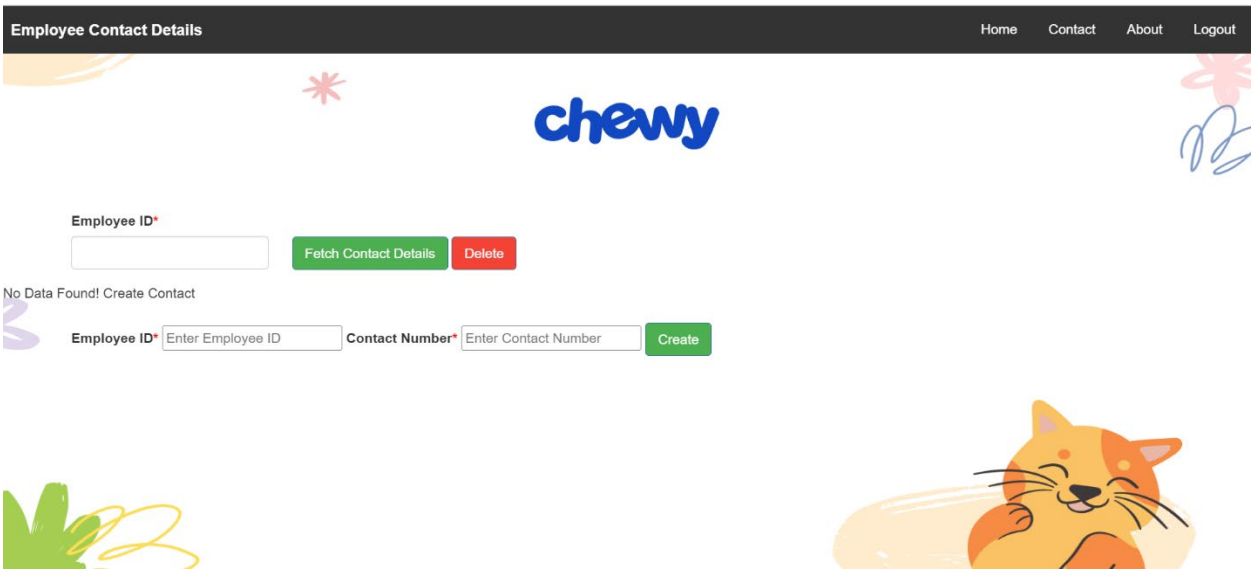
NAME	FOOD	BEDS	ACCESSORIES	CLOTHING	TOYS	GROOMING	TOTAL
Angelica Putman	1	2	0	1	2	0	6
James Johnson	1	0	1	0	0	1	3
Wilma Anderson	2	0	0	0	1	0	3
Charles Lopez	0	0	1	1	0	0	2
Madison Kim	0	0	1	0	1	0	2

Sales analytics: Gives us details of most sold product by quantity and by revenue generated

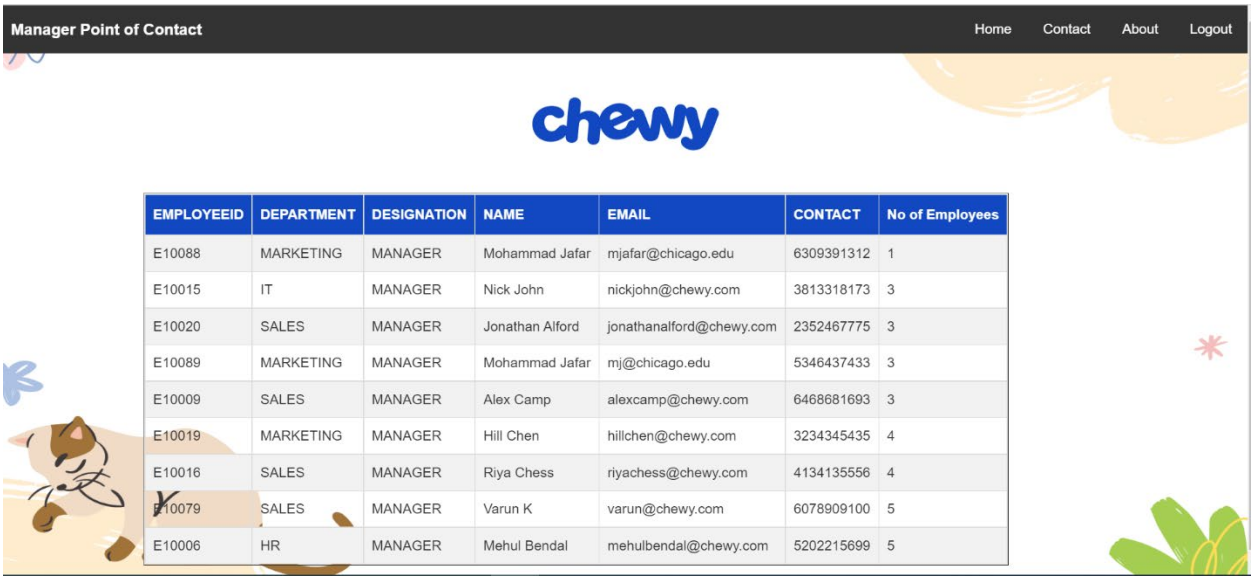


ITEM ID	BRAND	ITEM DESCRIPTION	PRICE PER ITEM	ITEM COUNT	RANK
I109	Barefoot	RoundBed	25	26	1
I104	Tuffy	ChewToy	10	24	2
I101	Nordog	DogTags	18	22	3
I112	Tuffy	RopeToy	11	22	4
I102	Nordog	DogHoodie	11	21	5

Employee contact details: Helps admin to update/delete/insert employee contact details



Manager information: On clicking CONTACT on nav-bar we get manager contact information for every department



EMPLOYEEID	DEPARTMENT	DESIGNATION	NAME	EMAIL	CONTACT	No of Employees
E10088	MARKETING	MANAGER	Mohammad Jafar	mjafar@chicago.edu	6309391312	1
E10015	IT	MANAGER	Nick John	nickjohn@chewy.com	3813318173	3
E10020	SALES	MANAGER	Jonathan Alford	jonathanalford@chewy.com	2352467775	3
E10089	MARKETING	MANAGER	Mohammad Jafar	mj@chicago.edu	5346437433	3
E10009	SALES	MANAGER	Alex Camp	alexcamp@chewy.com	6468681693	3
E10019	MARKETING	MANAGER	Hill Chen	hillchen@chewy.com	3234345435	4
E10016	SALES	MANAGER	Riya Chess	riyachess@chewy.com	4134135556	4
E10079	SALES	MANAGER	Varun K	varun@chewy.com	6078909100	5
E10006	HR	MANAGER	Mehul Bendal	mehulbendal@chewy.com	5202215699	5

Chapter 7: Implementation details

Database Hosting

The current database we have for Chewy uses Oracle SQL runs on the Eller Management Information Systems servers, which is free until the end of the academic year. Once the subscription ends for the academic year, our team recommends that Chewy uses Amazon Aurora Serverless for their database needs. Amazon Aurora Serverless will allow Chewy to seamlessly scale capacity up or down based on their eCommerce needs, connect other applications, and pay based on per-second usage. The benefits of using Amazon Aurora Serverless include its scalability, cost-effectiveness through cloud usage, transparency, and durability ("*Amazon Aurora Serverless*", 2022).

Chewy is an extremely large e-Commerce platform that receives about ~44 million users per month, or 1.45 million users daily ("SimilarWeb", 2022). The average customer visits 6 pages on

Chewy before leaving, and if we assume our PHP script requires 10 kb for each page a unique user visits, then each visitor would use ~ 60 kb. 1 terabyte [TB] (or 1,000 GB) = 1,073,741,824 kilobytes, so if we divide this number by 60 kb, 1 TB would provide enough data to handle 17,895,697 unique visitors. Therefore, Chewy can start with a 2,000 GB database at the beginning of the month and grow at 2%, or 40 GB daily, to effectively handle their 44 million monthly visitors. At the end of a 30-day period, the following usage in GB-month would be 2,580 GB-month. If we assume data is stored in the US West (Oregon) region, the unit cost is $0.10 per GB-month. Therefore, the total cost would be 2,580 GB-month * $0.10 per GB-month = $258 each month ("Amazon Aurora Pricing", 2022).

For I/O charges, we can assume that the Aurora database reads 100 pages from storage per second to read the queries operating on it. If we perform the monthly calculations, we found

- Days: 30
- Hours: 24
- Minutes: 60
- Seconds: 60
- Pages per second: 100

Total Monthly Requests: (100 pages per second * 60 seconds * 60 minutes * 24 hours * 30 days) → 259,200,000 (259.2 million)

I/O Rate: $0.20 per 1 million request if hosted in the US West (Oregon) region.

Total I/O Cost: $51.84

Therefore, Chewy would require $258 for database hosting and $51.84 on I/O operations every month. Total monthly costs would equate to ~ $310, which would equal $3,720 per year.

Front-End Hosting

The dashboard our team created uses PHP to allow Chewy admin to effectively navigate and understand different aspects of the business. There are no direct costs associated with the development of this dashboard, but we can assume that 3 PHP developers will be required to maintain this site and update its functionality as Chewy continues to grow. The average salary of

a PHP developer is $91,000 annually, so $273,000 would be incurred in people costs to maintain and expand upon the existing admin dashboard ("*PHP Developer Salary*", 2022).

Citations

Amazon Aurora Pricing. (2022). Retrieved December 6, 2022, from https://aws.amazon.com/rds/aurora/pricing/

Amazon Aurora Serverless. Amazon. (2022). Retrieved December 6, 2022, from https://aws.amazon.com/rds/aurora/serverless/

Chewy.com. SimilarWeb. (2022). Retrieved December 6, 2022, from https://www.similarweb.com/website/chewy.com/#overview

PHP Developer Salary (December 2022) - ZIPPIA | Average PHP developer ... PHP Developer Salary. (2022). Retrieved December 6, 2022, from https://www.zippia.com/php-developer-jobs/salary/

Appendix A

Learnings

Before we began working on this project, most of us did not have a very good knowledge of how a database operates, even though we all had previous job experience in a variety of fields.

We started off by conducting research on our customer Chewy. We noticed several issues with the system that is currently in place. Based on this, we devised a requirement analysis and formulated a detailed problem statement. We started working on the ER diagram as the class proceeded into the concepts that were being covered. Even though we created our ER with a larger scope in mind, we quickly learned that our client viewed things from two different angles: one was the administrative side, and the other was the customer-facing side. Due to the short amount of time, we restricted ourselves to working only on the administrative dashboard.

After we finished designing the ER, we translated the data into relational tables. In order to move forward, we normalized everything so that every table was in the 4NF form.

To create features such as the vaccination tracker and the apply discount functionality, amongst many others, we used a variety of complex SQL techniques like WITH, joins, and procedures. During the process of establishing tables and inserting data, we further made it a point to adhere to best practices by adding checks and constraints.

We simultaneously began working on the front end of the project because we needed to implement something that Chewy admin could easily use to implement our SQL queries for increased functionality plore that aspect as well. We developed our website using HTML, CSS, and PHP for server-side scripting on the front end.

In the end, we decided to host our website on an AWS EC 2 instance.

To summarize, we all learned about the entire database lifecycle from development to implementation & maintenance, as well as learned about database management and best practices for data integrity, security, and consistency.

Appendix B

Strong Entities Table scripts

```
-- Customers Table
DROP TABLE CUSTOMERS;
CREATE TABLE CUSTOMERS (
customerID char(6) constraint customer_pk primary key,
fName varchar(50) constraint fName_cus not null,
lName varchar(50),
custusername varchar(50) constraint custusername_cus not null,
custpassword varchar(15) constraint password_cus not null,
custEmail varchar(50) constraint custemail_cus not null,
state char(2) constraint state_cus not null, --can add check constraint
street varchar(50) constraint street_cus not null,
city varchar(50) constraint city_cus not null,
zip char(5) constraint zip_cus not null,
constraint username_cust UNIQUE (custusername)
constraint passcheck CHECK (REGEXP_COUNT(PASSWORD, '[0-9]') >= 1
```

```
        AND REGEXP_COUNT(PASSWORD, '[A-Z]') >= 1
        AND REGEXP_COUNT(PASSWORD, '[a-z]') >= 1
        AND LENGTH(PASSWORD) >= 8)
);

-- Customer Phones Table
DROP TABLE CUSTOMER_PHONES;

CREATE TABLE CUSTOMER_PHONES (
customerID char(6) NOT NULL,
custPhone char(10) NOT NULL,
FOREIGN KEY (customerID) REFERENCES CUSTOMERS (customerID),
constraint cust_phone_pk primary key (customerID, custPhone)
);
COMMIT;

-- Employees Table
DROP TABLE EMPLOYEES;
CREATE TABLE EMPLOYEES (
employeeID char(6) constraint employee_pk primary key,
password varchar(15) constraint password_nl not null,
fName varchar(50) constraint fName_nl not null,
mName varchar(50),
lName varchar(50),
employeeEmail varchar(50) constraint empemail_nl not null,
designation varchar(20) constraint design_nl not null,
department varchar(20) constraint deprt_nl not null,
managerid varchar(6),
ssn char(9) constraint ssn_un UNIQUE,
dateOfHire DATE DEFAULT sysdate NOT NULL, --can add default value
state char(2) constraint state_nl not null, --can add check constraint
street varchar(20) constraint street_nl not null,
```

```
city varchar(20) constraint city_nl not null,

zip char(5) constraint zipnl not null,

constraint empemail_un UNIQUE (employeeEmail),

constraint desig CHECK (designation IN ('ASSOCIATE', 'LEAD', 'MANAGER',
'CEO', 'CFO', 'CTO', 'CIO')),

constraint depart CHECK (department IN ('ADMIN', 'IT', 'SALES', 'HR',
'MARKETING')),

constraint passcheck CHECK (REGEXP_COUNT(PASSWORD, '[0-9]') >= 1

       AND REGEXP_COUNT(PASSWORD, '[A-Z]') >= 1

       AND REGEXP_COUNT(PASSWORD, '[a-z]') >= 1

       AND LENGTH(PASSWORD) >= 8)

);


-- Employee Phones Table

CREATE TABLE EMPLOYEE_PHONES
   (EMPLOYEEID VARCHAR2(6),
    EMPLOYEEPHONE VARCHAR2(30) CONSTRAINT EMPID_PK NOT NULL,
    constraint EMPID_PK primary key (EMPLOYEEID, EMPLOYEEPHONE),
    FOREIGN KEY (EMPLOYEEID) REFERENCES EMPLOYEES(EMPLOYEEID)
);


-- Adoption Associates Table

CREATE TABLE ADOTPTION_ASSOCIATES (

employeeID char(6) constraint employee_pk_adop primary key,

highestDegree varchar(20) NOT NULL,

degreeMajor varchar(20) NOT NULL,

availabiltyPercentage NUMBER NOT NULL,

constraint availPer CHECK (availabiltyPercentage <= 1),

FOREIGN KEY (employeeID) References EMPLOYEES (employeeID)

);


-- ADOPTION_ASSOCIATE_TRAINING

CREATE TABLE ADOPTION_ASSOCIATE_TRAINING (

employeeID char(6) NOT NULL,
```

```
trainingCertificateID char(7) NOT NULL,
constraint aaTraining_pk primary key (employeeID, trainingCertificateID),
FOREIGN KEY (employeeID) References ADOTPTION_ASSOCIATES (employeeID)
);

-- CUSTOMER_SUPPORT_AGENTS
CREATE TABLE CUSTOMER_SUPPORT_AGENTS (
employeeID char(6) constraint employee_pk_custSup primary key,
supportLocation varchar(20) NOT NULL,
FOREIGN KEY (employeeID) References EMPLOYEES (employeeID)
);

-- CUST_SUPPORT_AGENTS_LANGUAGES (multivalued attribute)
CREATE TABLE CUST_SUPPORT_AGENTS_LANGUAGES (
employeeID char(6) NOT NULL,
languageName varchar(20) NOT NULL,
proficiency varchar(15) NOT NULL,
constraint custSup_language_pk primary key (employeeID, languageName),
constraint exper CHECK (proficiency IN ('Novice', 'Intermediate', 'Advanced',
'Fluent')),
FOREIGN KEY (employeeID) References CUSTOMER_SUPPORT_AGENTS (employeeID)
);

-- CUST_SUPPORT_AGENTS_STATES (multivalued attribute)
CREATE TABLE CUST_SUPPORT_AGENTS_STATES (
employeeID char(6) NOT NULL,
stateCovered char(2) NOT NULL,
constraint custSup_state_pk primary key (employeeID, stateCovered),
FOREIGN KEY (employeeID) References CUSTOMER_SUPPORT_AGENTS (employeeID)
);

-- DEVELOPERS Table
```

```
CREATE TABLE DEVELOPERS (
employeeID char(6) constraint employee_pk_dev primary key,
hourlyRate NUMBER not null,
devSpecialization varchar(15) constraint devSpecial_nl not null,
FOREIGN KEY (employeeID) References EMPLOYEES (employeeID)
);

-- DEVELOPERS_CODING Table (multivalued attribute)
CREATE TABLE DEVELOPER_CODING (
employeeID char(6) NOT NULL,
techLanguageName varchar(20) NOT NULL,
proficiency varchar(15) NOT NULL,
constraint devCode_pk primary key (employeeID, techLanguageName),
constraint prof CHECK (proficiency IN ('Novice', 'Intermediate', 'Advanced',
'Master')),
FOREIGN KEY (employeeID) References DEVELOPERS (employeeID)
);

-- MANAGERS
CREATE TABLE MANAGERS (
employeeID char(6) constraint employee_pk_managers primary key,
highestDegree varchar(20) NOT NULL,
yearsOfManagement NUMBER NOT NULL,
constraint highDeg CHECK (highestDegree IN ('Masters', 'Doctoral')),
FOREIGN KEY (employeeID) References EMPLOYEES (employeeID)
);

-- INSURANCE AGENTS
CREATE TABLE INSURANCE_AGENTS (
employeeID char(6) constraint employee_pk_insurance primary key,
agentCertification varchar(20),
agentLicense varchar(20),
```

```
yearsOfExp NUMBER NOT NULL,
FOREIGN KEY (employeeID) References EMPLOYEES (employeeID)
);

-- VETERINARIANS Table
CREATE TABLE VETERINARIANS (
employeeID char(6) constraint employee_pk_vets primary key,
vetSpecialization varchar(15) constraint special_nl not null,
clinicState char(2) constraint clin_state_nl not null, --can add check
constraint
clinicStreet varchar(20) constraint clin_street_nl not null,
clinicCity varchar(20) constraint clin_city_nl not null,
clinincZip char(5) constraint clin_zipnl not null,
FOREIGN KEY (employeeID) References EMPLOYEES (employeeID)
);

-- VETERINARIANS_CERTIFICATES Table (multivalued attribute)
CREATE TABLE VETERINARIANS_CERTIFICATES (
employeeID char(6) NOT NULL,
vetCertificateID char(7) NOT NULL,
constraint vetCert_pk primary key (employeeID, vetCertificateID),
FOREIGN KEY (employeeID) References VETERINARIANS (employeeID)
);

-- Insurance Table
CREATE TABLE INSURANCE(
POLICYID CHAR(6),
PREMIUMAMOUNT NUMBER,
POLICYDATE DATE,
POLICYDESCRIPTION VARCHAR2(30),
POLICYNAME VARCHAR2(30),
DUEDATE DATE,
```

```
CONSTRAINT POLICYID_PK PRIMARY KEY (POLICYID)
);


-- Inventory Table
CREATE TABLE INVENTORY (
inventoryID char(5) constraint inventory_pk primary key,
inventoryName varchar(40) constraint inventoryName_c1 not null
);

-- Invoice Table
CREATE TABLE INVOICE (
invoiceID char(6) constraint invoice_pk primary key,
status varchar(20),
paymentMode varchar(20),
state char(2) constraint state_inv not null, --can add check constraint
street varchar(50) constraint street_inv not null,
city varchar(50) constraint city_inv not null,
zip char(5) constraint zip_inv not null,
orderID char(6),
FOREIGN KEY (orderID) REFERENCES ORDERS(orderID),
constraint invoicestatuscheck CHECK (status IN ('CLOSED', 'OPEN', 'Closed',
'Open'))
);

-- Items Table
CREATE TABLE ITEMS (
itemID varchar(4) constraint itemID_pk primary key,
itemDesc varchar(20) constraint itemDesc_c1 NOT NULL,
price NUMBER(5,2) constraint price_c1 NOT NULL,
discountedPrice NUMBER(5,2),
brand varchar(15),
```

```
prodTypeID char(5) not null,
FOREIGN KEY (prodTypeID) REFERENCES PRODUCT_TYPES (prodTypeID)
);

-- Shopping Products Table
CREATE TABLE SHOPPING_PRODUCTS (
ITEMID VARCHAR2(20) NOT NULL ENABLE,
COLOR VARCHAR2(64) NOT NULL ENABLE,
SIZEPRODUCTCATEGORY VARCHAR2(64),
CONSTRAINT ITEMID2_PK PRIMARY KEY (ITEMID),
FOREIGN KEY (ITEMID) REFERENCES ITEMS (ITEMID)
);

-- Pharmacy Products Table
CREATE TABLE PHARMACY_PRODUCTS (
ITEMID VARCHAR2(20) NOT NULL,
DOSAGEPERDAY VARCHAR2(64) NOT NULL,
DOSAGETIME NUMBER(4,0),
TREATMENTTYPE VARCHAR2(64),
CONSTRAINT ITEMID3_PK PRIMARY KEY (ITEMID)
);

-- Local Shelter Table
CREATE TABLE LOCAL_SHELTERS (
SHELTERID CHAR(6),
SHELTERNAME VARCHAR2(30),
CONTACTNAME VARCHAR2(30),
CONTACTEMAIL VARCHAR2(30),
CONTACTNUM NUMBER(30,0),
DUEDATE DATE,
STATE VARCHAR2(30),
STREET VARCHAR2(30),
```

```
CITY VARCHAR2(30),
ZIP VARCHAR2(30),
CHECK (contactemail LIKE '%@%.%' AND contactemail NOT LIKE '@%' AND
contactemail NOT LIKE '%@%@%'   ),
CHECK (contactnum NOT LIKE '%[^0-9]%'),
CONSTRAINT SHELTERIDPK PRIMARY KEY (SHELTERID)
);

-- Logins Table
CREATE TABLE LOGINS (
USERNAME VARCHAR2(20) NOT NULL,
PASSWORD VARCHAR2(64) NOT NULL,
EMPLOYEEID CHAR(6) NOT NULL,
CONSTRAINT EMPID3_PK PRIMARY KEY (EMPLOYEEID),
FOREIGN KEY (EMPLOYEEID) REFERENCES EMPLOYEES (EMPLOYEEID)
);

-- Orders Table
CREATE TABLE ORDERS (
orderID char(6) constraint order_pk primary key,
orderDate DATE DEFAULT sysdate NOT NULL,
deliveryPrice number(5,0),
status varchar(20),
customerID char(6),
FOREIGN KEY (customerID) REFERENCES CUSTOMERS(CustomerID),
constraint statuscheck CHECK (status IN ('CLOSED', 'OPEN', 'Closed', 'Open'))
);

-- Pets Table
DROP TABLE PETS;
CREATE TABLE PETS (
petID char(6) constraint pet_pk primary key,
```

```
petName varchar(20) not null,
petType varchar(15) not null,
petBirthday DATE,
petAge NUMBER(5,2),
adoptionDate DATE,
petBreed varchar(15) not null,
petWeight NUMBER(3),
customerID char(6),
FOREIGN KEY (CUSTOMERID) REFERENCES CUSTOMERS(CUSTOMERID)
);

-- Product Type Table
DROP TABLE PRODUCT_TYPE;
CREATE TABLE PRODUCT_TYPES (
prodTypeID char(5) constraint product_pk primary key,
prodType varchar(25) constraint prodType_c1 not null
);

-- Tickets Table
CREATE TABLE TICKETS (
ticketID char(6) constraint tickets_pk primary key,
department varchar(20) constraint deprts_nl not null,
ticketDesc varchar(50),
ticketStart timestamp(2) not null,
ticketEnd timestamp(2),
processingTime INTERVAL DAY TO SECOND AS (ticketEnd - ticketStart),
ticketPriority char(10) constraint ticketsPri_nl not null,
ticketStatus varchar(10) constraint status_c1 not null,
constraint departs CHECK (department IN ('VET SERVICES', 'IT', 'SALES', 'HR',
'MARKETING')),
constraint tixPri CHECK (ticketPriority IN ('Low', 'Medium', 'High'))
);
```

```
-- Training Course Table
CREATE TABLE TRAINING_COURSES (
courseID char(6) constraint course_pk primary key,
courseDesc varchar(50),
courseStartDate DATE DEFAULT sysdate NOT NULL,
courseDuration varchar(10),
employeeID char(6),
FOREIGN KEY (EMPLOYEEID) REFERENCES EMPLOYEES(EMPLOYEEID)
);

-- Vaccination Types Table
CREATE TABLE VACCINATION_TYPES (
vacID varchar(5) constraint vacID_pk primary key,
vacName varchar(30) constraint vacName_c1 NOT NULL
constraint vacName_c2 CHECK (vacName IN ('DAPP', 'Rabies-Dogs', 'Rabies-
Cats', 'FVRCP', 'Rabies-Cats-Booster', 'Rabies-Dogs-Booster'))
);

-- Vendors
CREATE TABLE VENDORS (
vendorID char(6) constraint vendor_pk primary key,
vendorName varchar(50) NOT NULL,
contactName varchar2(30),
contactEmail varchar2(30) check (contactemail LIKE '%@%.%' AND contactemail
NOT LIKE '@%' AND contactemail NOT LIKE '%@%@%'   ),
contactNum number(30) CHECK (contactnum NOT LIKE '%[^0-9]%'),
state char(2) constraint vState_nl not null,
street varchar(20) constraint vStreet_nl not null,
city varchar(20) constraint vCity_nl not null,
zip char(5) constraint vZipnl not null
);
```

```
-- Vendor Types Table
CREATE TABLE VENDOR_TYPES (
vtypeID char(7) constraint vendortype_pk primary key,
vtypeName varchar(50) NOT NULL
);

-- Vet Services Table
DROP TABLE VET_SERVICES;
CREATE TABLE VET_SERVICES (
serviceID char(6) constraint service_pk2 primary key,
serviceType varchar(20),
serviceLocation varchar(20),
serviceOrderDate DATE DEFAULT sysdate NOT NULL,
orderID char(6),
FOREIGN KEY (orderID) REFERENCES ORDERS(orderID),
constraint vs CHECK (serviceType IN ('Vaccinations', 'Parasite Control',
'Grooming', 'Dental'))
);

-- DENTAL
CREATE TABLE DENTAL (
serviceID char(6) constraint service_dental_pk primary key,
procedureType varchar(50),
prescription varchar(50),
FOREIGN KEY (serviceID) REFERENCES VET_SERVICES(serviceID)
);

-- DENTAL_TOOlS_USED (multivalued attribute)
CREATE TABLE DENTAL_TOOlS_USED (
serviceID char(6) NOT NULL,
tool varchar(15) NOT NULL,
constraint dentalTool_pk primary key (serviceID, tool),
```

```
FOREIGN KEY (serviceID) References DENTAL (serviceID)
);

-- Grooming Products Table
CREATE TABLE GROOMING_PRODUCTS (
serviceID char(6) NOT NULL,
productsUsed varchar2(70) NOT NULL,
constraint grooming_products_pk primary key (serviceID, productsUsed),
FOREIGN KEY (serviceID) References VET_SERVICES(serviceID)
);

-- Grooming Type Table
CREATE TABLE GROOMING_TYPE (
serviceID char(6) NOT NULL,
groomingtype varchar2(70) NOT NULL,
constraint grooming_type_pk primary key (serviceID, groomingtype),
FOREIGN KEY (serviceID) References VET_SERVICES(serviceID)
);

-- PARASITE_CONTROL
CREATE TABLE PARASITE_CONTROL (
serviceID char(6) constraint service_parasite_pk primary key,
parasiteType varchar(20),
medicine varchar(50),
dose number(2),
FOREIGN KEY (serviceID) REFERENCES VET_SERVICES(serviceID)
);

-- Vaccination Table
DROP TABLE VACCINATIONS;
CREATE TABLE VACCINATIONS (
serviceID char(4) constraint service_pk primary key,
```

```
vacID varchar(5) constraint vacID_c1 not null,
vacDate DATE DEFAULT sysdate NOT NULL,
nextDue DATE DEFAULT sysdate
);
```

Weak Entities & Relationships Table scripts

```
-- ASSOCIATES_CONNECTS_SHELTERS
CREATE TABLE ASSOCIATES_CONNECTS_SHELTERS (
shelterID char(6) NOT NULL,
employeeID char(7) NOT NULL,
constraint assoc_shelter_pk primary key (shelterID, employeeID),
FOREIGN KEY (employeeID) References ADOTPTION_ASSOCIATES (employeeID),
FOREIGN KEY (shelterID) References LOCAL_SHELTERS (shelterID)
);

-- CAN_SUPPLY
CREATE TABLE CAN_SUPPLY (
vendorID char(6) NOT NULL,
prodTypeID char(5) NOT NULL,
constraint can_supply_pk primary key (vendorID, prodTypeID),
FOREIGN KEY (vendorID) References VENDORS (vendorID),
FOREIGN KEY (prodTypeID) References PRODUCT_TYPES (prodTypeID)
);

-- INSURANCE_PROVIDED
CREATE TABLE INSURANCE_PROVIDED (
employeeID char(6) NOT NULL,
policyID char(6) NOT NULL,
constraint insurance_provided_pk primary key (employeeID, policyID),
FOREIGN KEY (employeeID) References INSURANCE_AGENTS (employeeID),
FOREIGN KEY (policyID) References INSURANCE (policyID)
```

```
);

-- INVENTORY_DETAILS
CREATE TABLE INVENTORY_DETAILS (
itemID varchar2(4) NOT NULL,
inventoryID char(5) NOT NULL,
itemQuantity number(5) NOT NULL,
haveSupply char(1) Not NULL,
constraint inventory_details_pk primary key (itemID, inventoryID),
FOREIGN KEY (itemID) References ITEMS(itemID),
FOREIGN KEY (inventoryID) References INVENTORY(inventoryID)
);

-- ITEM_DETAILS
DROP TABLE item_details;
CREATE TABLE ITEM_DETAILS (
orderID char(6) not null,
itemID varchar(4) not null,
itemQuantity NUMBER(5,2) default 1,
FOREIGN KEY (orderID) REFERENCES ORDERS (orderID),
FOREIGN KEY (itemID) REFERENCES ITEMS (itemID),
constraint order_item_pk primary key (orderID, itemID)
);

-- SERVICE_DETAILS
CREATE TABLE SERVICE_DETAILS (
employeeID char(6) NOT NULL,
serviceID char(6) NOT NULL,
serviceStartTime varchar(50) NOT NULL,
serviceEndTime varchar(50) NOT NULL,
serviceDuration varchar(50) NOT NULL,
```

```
constraint service_details_pk primary key (employeeID, serviceID,
serviceStartTime),
FOREIGN KEY (employeeID) references VETERINARIANS (employeeID),
FOREIGN KEY (serviceID) References VET_SERVICES(serviceID)
);

-- VENDOR_CONTENTS
CREATE TABLE VENDOR_CONTENTS(
vendorID char(6) NOT NULL,
vtypeID char(7) NOT NULL,
constraint vendor_content_pk primary key (vendorID, vtypeID),
FOREIGN KEY (vendorID) References VENDORS(vendorID),
FOREIGN KEY (vtypeID) References VENDOR_TYPES(vtypeID)
);
```

Sequence & Trigger Scripts

```
-- Course Sequence
CREATE SEQUENCE course_seq
    START WITH 10000
    MAXVALUE 99999;

CREATE OR REPLACE TRIGGER course_id_gen
  BEFORE INSERT
  ON TRAINING_COURSES
  FOR EACH ROW
BEGIN
       :new.employeeID := 'C' || course_seq.NEXTVAL;
END;
/

-- Customer ID Sequence & Trigger
```

```
DROP SEQUENCE cust_seq;
CREATE SEQUENCE cust_seq
    START WITH 10000
    MAXVALUE 99999;

CREATE OR REPLACE TRIGGER cust_id_gen
  BEFORE INSERT
  ON CUSTOMERS
  FOR EACH ROW
BEGIN
       :new.customerID := 'C' || cust_seq.NEXTVAL;
END;
/

-- Employee ID Sequence & Trigger
DROP SEQUENCE employees_seq;
CREATE SEQUENCE employees_seq
    START WITH 10000
    MAXVALUE 99999;

CREATE OR REPLACE TRIGGER emp_id_gen
  BEFORE INSERT
  ON EMPLOYEES
  FOR EACH ROW
BEGIN
       :new.employeeID := 'E' || employees_seq.NEXTVAL;
END;
/

-- Insurance ID Sequence & Trigger
CREATE SEQUENCE insurance_seq
    START WITH 100
```

```
    MAXVALUE 999;
CREATE OR REPLACE TRIGGER policy_id_gen
  BEFORE INSERT
  ON INSURANCE
  FOR EACH ROW
BEGIN
       :new.policyID := 'P' || insurance_seq.NEXTVAL;
END;
/

-- Inventory Sequence & Trigger
DROP SEQUENCE inventory_seq;
CREATE SEQUENCE inventory_seq
    START WITH 100
    MAXVALUE 999;

CREATE OR REPLACE TRIGGER inventory_id_gen
  BEFORE INSERT
  ON INVENTORY
  FOR EACH ROW
BEGIN
       :new.inventoryID := 'IN' || inventory_seq.NEXTVAL;
END;
/

-- Invoice Sequence & Trigger
CREATE SEQUENCE invoice_seq
    START WITH 10000
    MAXVALUE 99999;

CREATE OR REPLACE TRIGGER invoice_id_gen
```

```
  BEFORE INSERT
  ON INVOICE
  FOR EACH ROW
BEGIN
       :new.invoiceID := 'I' || orders_seq.NEXTVAL;
END;
/

-- Item ID Sequence & Trigger
DROP SEQUENCE item_seq;
CREATE SEQUENCE item_seq
    START WITH 100
    MAXVALUE 999;

CREATE OR REPLACE TRIGGER item_id_gen
  BEFORE INSERT
  ON ITEMS
  FOR EACH ROW
BEGIN
       :new.itemID := 'I' || item_seq.NEXTVAL;
END;
/

-- Orders Sequence & Trigger
DROP SEQUENCE orders_seq;
CREATE SEQUENCE orders_seq
    START WITH 10000
    MAXVALUE 99999;

CREATE OR REPLACE TRIGGER order_id_gen
  BEFORE INSERT
  ON ORDERS
```

```
  FOR EACH ROW
BEGIN
       :new.orderID := 'O' || orders_seq.NEXTVAL;
END;
/

-- Pet ID Sequence & Trigger
DROP SEQUENCE pet_seq;
CREATE SEQUENCE pet_seq
    START WITH 10000
    MAXVALUE 99999;

CREATE OR REPLACE TRIGGER pet_id_gen
  BEFORE INSERT
  ON PETS
  FOR EACH ROW
BEGIN
       :new.petID := 'P' || pet_seq.NEXTVAL;
END;
/

-- Product Type Sequence & Trigger
DROP SEQUENCE product_seq;
CREATE SEQUENCE product_seq
    START WITH 100
    MAXVALUE 999;

CREATE OR REPLACE TRIGGER product_id_gen
  BEFORE INSERT
  ON PRODUCT_TYPE
  FOR EACH ROW
BEGIN
```

```
       :new.prodTypeID := 'PT' || product_seq.NEXTVAL;
END;
/


-- Shelter ID Sequence & Trigger
CREATE SEQUENCE shelter_seq
    START WITH 100
    MAXVALUE 999;

CREATE OR REPLACE TRIGGER shelter_id_gen
  BEFORE INSERT
  ON LOCAL_SHELTERS
  FOR EACH ROW
BEGIN
       :new.shelterID := 'LS' || shelter_seq.NEXTVAL;
END;
/


-- Ticket ID Sequence & Trigger
CREATE SEQUENCE ticket_seq
START WITH 10000
MAXVALUE 99999;

CREATE OR REPLACE TRIGGER ticket_id_gen
BEFORE INSERT
ON TICKETS
FOR EACH ROW
BEGIN
    :new.ticketID := 'T' || ticket_seq.NEXTVAL;
END;
/
```

```oracle_plsql
-- Training Certificate Sequence & Trigger
CREATE SEQUENCE trainCert_seq
    START WITH 10000
    MAXVALUE 99999;

CREATE OR REPLACE TRIGGER trainCert_id_gen
  BEFORE INSERT
  ON ADOPTION_ASSOCIATE_TRAINING
  FOR EACH ROW
BEGIN
       :new.trainingCertificateID := 'AT' || trainCert_seq.NEXTVAL;
END;
/


-- Vendor ID Sequence & Trigger

CREATE SEQUENCE vendor_seq

    START WITH 10000

    MAXVALUE 99999;


CREATE OR REPLACE TRIGGER vendorid_gen

  BEFORE INSERT

  ON VENDORS

  FOR EACH ROW

BEGIN

       :new.vendorID := 'V' || vendor_seq.NEXTVAL;

END;

/


-- Vendor Type ID Sequence & Trigger

CREATE SEQUENCE vendorType_seq

    START WITH 10000

    MAXVALUE 99999;
```

```
CREATE OR REPLACE TRIGGER vendorTypeid_gen
  BEFORE INSERT
  ON VENDOR_TYPES
  FOR EACH ROW
BEGIN
       :new.vtypeID := 'VT' || vendorType_seq.NEXTVAL;
END;
/


-- Vet Certificate Sequence & Trigger
CREATE SEQUENCE vetCert_seq
    START WITH 10000
    MAXVALUE 99999;

CREATE OR REPLACE TRIGGER vetCert_id_gen
  BEFORE INSERT
  ON VETERINARIANS_CERTIFICATES
  FOR EACH ROW
BEGIN
       :new.vetCertificateID := 'VC' || vetCert_seq.NEXTVAL;
END;
/


-- Vet Services ID Sequence & Trigger
DROP SEQUENCE vetservices_seq;
CREATE SEQUENCE vetservices_seq
    START WITH 100
    MAXVALUE 999;
CREATE OR REPLACE TRIGGER service_id_gen
  BEFORE INSERT
  ON VET_SERVICES
  FOR EACH ROW
BEGIN
       :new.serviceID := 'S' || vetservices_seq.NEXTVAL;
```

```
END;
/
```