Animal Shelter Database System Project

Project Objective:

Organizing tons of data in every organization is important to effectively manage operations and streamline processes. QUELIFE is an animal shelter organization for which we will be designing a database system. This organization has multiple branches spread across Wisconsin, Illinois, Missouri, Montana, California, Texas, and Nevada. This database will encompass primary details related to facilities, animal descriptions and transfer details, sponsor details and employee details. We are also ensuring to maintain high quality of the database using normalization.

Requirement Analysis:

We are analysing the requirements to setup a database for this organization. We have multiple entities to consider along with their relationships. Here we have multiple branches of shelter at multiple locations with various facilities. Animal related records including animal information, animal features, adoption records and surrender details are need as well as the history of each animal. We are enabling the option to transfer of animals from one branch to another. Employee details, sponsor details and payment details are required to track the working of the shelter. Applying normalization to the tables and splitting the tables to ensure that there is no information loss.

Assumptions:

1. Animals can be transferred from one branch to another
2. Each shelter branch will have multiple employees
3. Each shelter branch will have a branch manager
4. Employees work only in one shelter branch only (no transfer of employees)
5. An animal can be adopted/surrendered to the shelter for various reasons multiple times
6. A vaccination is given each time an animal is detected with any disease
7. Sponsors can support multiple shelter branches
8. Full payment must be done in a single payment for adoption of an animal
9. Each medical condition of an animal is treated by only one veterinarian
10. Veterinarians can treat multiple conditions of multiple animals
11. Each transfer of an animal involves transfer from one branch to another
12. Each veterinary doctor can have multiple specializations
13. All contacts should not contain spaces
14. Employees work on hourly basis and hourly pay varies from employee to employee
15. Any animal is considered as ready for adoption if its vaccination status is up to date
16. Gross pay is the total pay for an employee without any deduction
17. Net pay is the pay of an employee after tax deduction
18. Employee salary is an estimate considering the maximum number of hours they can work
19. Discount is applied on adoption fee if the amount is greater than 750 by 5% and when 1000 by 10%
20. Surrender reason cannot exceed 20 characters
21. Date of birth of an animal cannot be greater than date of acquisition
22. Employee department is selected depending on their education only
23. Employee has a unique hourly pay until unless the pay is changed and the table is updated
24. Each breed determines the type of animal
25. Address of the shelter has an unique pincode

Updated Entities and Relationships:

1. **shelter_branch:**
    - Entity Description: This entity contains details about the geographic location and details of each branch
    - Attributes: shelter_name, shelter_loc, address, shelter_contact
    - Relationships:
        - One-to-one relationship with facilities
        - Many-to-many relationship with animal_info
        - One-to-many relationship with employees_staff_details
        - Many-to-many relationship with animal_transfer_records
        - Many-to-many relationship with sponsor
        - Many-to-one relationship with shelter_loc_details

2. **shelter_loc_details (new table)**
    - Entity Description: This entity contains details about the shelter location of shelter
    - Attributes: shelter_loc, pin_code
    - Relationships:
        - One-to-many relationship with shelter_branch

3. **facilities:**
    - Entity Description: This entity represents amenities operated by each branch.
    - Attributes: fid, f_capacity, f_ani_count, f_area, play_areas, temp_control, outdoor_space
    - Relationships:
        - One-to-one relationship with shelter_branch

4. **animal_info:**
    - Entity Description: This entity contains information about the animal.
    - Attributes: animal_id, a_breed, a_dob ,date_of_acquisition, a_vaccine_status, animal_features_id
    - Relationships:
        - Many-to-many relationship with shelter_branch (as we have transfer facility)
        - One-to-one relationship with animal_features
        - Many-to-many relationship with surrender_details
        - Many-to-many relationship with adoption_details
        - One-to-many relationship with animal_transfer_records
        - Many-to-many relationship with nutrition
        - Ternary relationship with history of the animal which includes: grooming history, training history and medical history
        - Many-to-one relationship with *animal_breed_info*
5. ***animal_breed_info (new table)***
    - Entity Description: This entity contains information about the animal breed.
    - Attributes: a_breed, a_type
    - Relationships:
        - One-to-many relationship with animal_info
6. **animal_features:**
    - Entity Description: This entity contains physical attributes of the animal.
    - Attributes: animal_id, a_color, a_weight, a_height, a_eye_color, a_condition

- Relationships:
  - One-to-one relationship with animal_info

7. **adoption_details:**
   - Entity Description: This entity contains information about the adoption of an animal
   - Attributes: ad_ID, animal_id, ad_name, ad_contact, ad_date, ad_fee.
   - Relationships:
     - Many-to-many relationship with animal_info
     - One-to-one relationship with payment_details

8. **surrender_details:**
   - Entity Description: This entity contains details about returning of the animal to the shelter
   - Attributes: s_id, s_date, s_name (person who brings animals), s_reason.
   - Relationships:
     - Many-to-many relationship with animal_info

9. **nutrition:**
   - Entity Description: This contains information about diet and nutrition of each animal
   - Attributes: food_id, food_type, food_quantity
   - Relationships:
     - Many-to-many relationship with animal_info

10. **training_history:**
    - Entity Description: This entity contains details about training the animals
    - Attributes: training_ID, trainer_name, training_date, training_type, training_stage.
    - Relationships:
      - Many-to-one relationship with animal_info
      - Many-to-many relationship with employees_staff_details

11. **grooming_history:**
    - Entity Description: This entity contains details about grooming the animals
    - Attributes: grooming_id, groomer_name, grooming_date, grooming_type.
    - Relationships:
      - Many-to-one relationship with animal_info
      - Many-to-many relationship with employees_staff_details

12. **medical_history:**
    - Entity Description: This entity contains details about the medical history of each animal
    - Attributes: medical_id, disease, treatment_date
    - Relationships:
      - Many-to-one relationship with animal_info
      - Many-to-many relationship with vet_id
      - One-to-one relationship between medical_history and vaccination
      - Many-to-one relationship with disease_mitigation

13. **disease_mitigation** ***(new table)***
    - Entity Description: This entity contains details about the disease_mitigation of each animal.
    - Attributes: disease, vaccination_id
    - Relationships:
        - One-to-many relationship with medical_history
        - One-to-one relationship between disease_mitigation and vaccination

14. **vaccination:**
    - Entity Description: This entity contains details about the vaccination of each animal
    - Attributes: vaccination_id, vaccination_name, vaccination_date, vaccination_dosage
    - Relationships:
        - One-to-one relationship between medical_history and vaccination

15. **employees_staff_details:**
    - Entity Description: This entity contains details about each worker or employee at the animal shelter QUELIFE
    - Attributes: emp_id, emp_name, emp_contact, emp_training, emp_education, emp_criminal_record
    - Relationships:
        - Many-to-one relationship with shelter_branch
        - Many-to-many relationship with grooming_history
        - Many-to-many relationship with training_history
        - Unary relationship – manages – branch manager
        - Many-to-one relationship with employees_stage

16. **employees_stage** ***(new table)***
    - Entity Description: This entity contains details about each employee stage at the animal shelter QUELIFE
    - Attributes: emp_level , emp_department, emp_education
    - Relationships:
        - One-to-many relationship with employees_staff_details

17. **sponsor:**
    - Entity Description: This entity contains details about sponsors given to shelter_branch
    - Attributes: sponsor_id, sponsor_name, sponsor_contact, sponsorship_type.
    - Relationships:
        - Many-to-many relationship with shelter_branch
        - One-to-many relationship with payment_details

18. **payment_details:**
    - Attributes: payment_ID, amount, payment_Date, payment_type.
    - Relationships:
        - One-to-one relationship with adoption_details
        - Many-to-one relationship with sponsor

19. **veterinary:**

- Entity Description: This entity contains details of the veterinarian
- Attributes: vet_id, vet_name, vet_contact, education
- Relationships:
  - Many-to-many relationship with medical_history
  - Many-to-one relationship with vet_edu

20. **Vet_edu*(new table):***
    - Entity Description: This entity contains details of the veterinarian education
    - Attributes: vet_name, vet_contact, work_exp
    - Relationships:
      - One-to-many relationship with veterinary
21. **animal_transfer_records:**
    - Entity Description: This entity contains details of transfer of animals from one branch to another
    - Attributes: transfer_id, transfer_date, transfer_from, transfer_to.
    - Relationships:
      - Many-to-one relationship with animal_info
      - Many-to-many relationship with shelter_branch
22. **employee_payroll:**
    - Entity Description: This entity contains details of the working hours of each employee.
    - Attributes: swipe_id, swipe_date, check_in_time , check_out_time, emp_id
    - Relationships:
      - Many-to-one relationship with employees_staff_details
      - Many-to-one relationship with employee_pay
23. **employee_pay*(new table):***
    - Entity Description: This entity contains details of the hourly pay of each employee.
    - Attributes: emp_id, hourly_pay
    - Relationships:
      - One-to-many relationship with employee_pay
      - Many-to-one relationship with employees_staff_details

UPDATED SCHEMA: (Primary Key: Underlined, Foreign Key: Italics)

shelter_branch (shelter_id, shelter_name, shelter_loc, address, shelter_contact)

animal_info (animal_id , a_breed, a_dob, date_of_acquisition, a_vaccine_status, *animal_features_id*)

animal_features (animal_features_id, a_color, a_weight, a_height, a_eye_color, a_condition)

animal_transfer_records (transfer_id, transfer_date, transfer_from, transfer_to, *animal_id*)

grooming_history (grooming_id, groomer_name, grooming_date, grooming_type)

nutrition (food_id, food_type, food_quantity)

sponsor (sponsor_id, sponsor_name, sponsor_contact, sponsorship_type)

surrender_details (s_id, s_date, s_name, s_reason)

training_history (training_ID, trainer_name, training_date, training_type, training_stage)

veterinary (vet_id, vet_name, vet_contact, education)

vet_specialization(*vet_id*, specialization)

medical_history (medical_id, disease, treatment_date)

payment_details (payment_ID, *sponsor_id* , amount, payment_Date, payment_type)

employees_staff_details (emp_id, emp_name, emp_contact, emp_training, emp_education, emp_criminal_record)

facilities (facilities_id, *shelter_id* , f_capacity, f_ani_count, f_area, play_areas, temp_control, outdoor_space)

adoption_details (ad_ID, *payment_ID*, ad_name, ad_contact, ad_date, ad_fee)

vaccination (vaccination_id, vaccination_name, vaccination_date, vaccination_dosage)

employee_payroll (swipe_id, *emp_id*, swipe_date, check_in_time, check_out_time)

perform *(grooming_id, emp_id)*

examines *(vet_id, medical_id)*

support *(shelter_id, sponsor_id)*

contain *(animal_id, ad_ID)*

need *(food_id, animal_id)*

submit *(s_id, animal_id)*

accommodate *(shelter_id, animal_id)*

transfers *(shelter_id, transfer_id)*

done_by *(training_ID, emp_id)*

stores (animal_id, training_history_id, grooming_history_id, medical_history)

shelter_loc_details (shelter_loc, pin_code)

animal_breed_info (a_breed, a_type)

disease_mitigation (disease, vaccination_id)

employees_stage(emp_level, emp_department, emp_education)

vet_edu (vet_name, vet_contact, work_exp)

employee_pay (*emp_id*, hourly_pay)

Database Normalization (Part 6):

Database normalization is necessary to design high quality tables. This step is necessary to maintain redundancy of the tables in such a way that no dependant information is lost due to deletion of any data rows. The normalize the tables in the database it is necessary to identify the functional dependencies in each table. Here is a list of functional dependencies we have identified in each table:

1. **shelter_branch:**
   shelter_id -> shelter_name, shelter_loc (shelter_id uniquely determines shelter_name and shelter_loc)

   shelter_id, shelter_name -> address, shelter_contact (For each tuple of shelter_ID and shelter_name, there is a unique address and contact)

   shelter_loc -> pin_code (shelter_loc uniquely determines pin_code)

2. **facilities:**
   facilities_id -> f_capacity, f_ani_count, temp_control, outdoor_space, shelter_id (shelter_id uniquely determines shelter_name and shelter_loc)

   facilities_id, outdoor_space -> f_area, play_areas

3. **animal_info:**
   animal_id -> a_type, a_breed, a_dob, date_of_acquisition, a_vaccine_status
   animal_id, a_type -> animal_features_id
   a_breed -> a_type

4. **animal_features:**
   animal_features_id -> a_color, a_weight, a_height, a_condition, a_eye_color
   animal_features_id, a_condition -> a_color, a_weight, a_height

5. **adoption_details:**
   ad_ID -> ad_name, ad_contact, ad_date, ad_fee, payment_ID
   ad_ID, ad_date -> payment_ID, ad_name, ad_contact

6. **surrender_details:**
   s_id -> s_date, s_name

s_id, s_date -> s_reason

7. **nutrition:**
   food_id -> food_type
   food_id, food_type -> food_quantity

8. **training_history:**
   training_ID -> trainer_name, training_date, training_type
   training_ID, training_date -> training_stage

9. **grooming_history:**
   grooming_id -> groomer_name, grooming_date
   grooming_id, groomer_name -> grooming_type

10. **medical_history:**
    medical_id -> disease, vaccination_id
    medical_id, disease -> vaccination_id, treatment_date

11. **vaccination:**
    vaccination_id -> vaccination_name, vaccination_date
    vaccination_id, vaccination_name -> vaccination_dosage

12. **employees_staff_details:**
    emp_id -> emp_name, emp_contact, emp_training, emp_education, emp_criminal_record
    emp_id, emp_contact -> emp_criminal_record
    emp_education -> emp_level
    emp_education -> emp_department

13. **sponsor:**
    sponsor_id -> sponsor_name, sponsor_contact
    sponsor_id, sponsor_name, sponsor_contact -> sponsorship_type

14. **payment_details:**
    payment_ID -> sponsor_id, payment_Date, payment_type
    payment_ID, sponsor_id -> amount, payment_Date, payment_type

15. **veterinary:**
    vet_id -> vet_name, vet_contact, education, work_exp
    vet_name, vet_contact -> work_exp

16. **animal_transfer_records:**
    transfer_id -> transfer_date, animal_id
    transfer_id, animal_id -> transfer_date, transfer_from, transfer_to

17. **employee_payroll:**
    swipe_id -> swipe_date, check_in_time, check_out_time, emp_id
    emp_id -> hourly_pay

Considering the FDs mentioned above, we are now checking if the tables are normalized. To normalize database system into BCNF preserving dependencies and maintaining lossless join, we are analysing each table and then decomposing into new tables wherever necessary. We can say that a schema or table is in BCNF if all the functional dependencies for that table

1. **shelter_branch:**
   shelter_id -> shelter_name, shelter_loc:
   Proof:
   This functional dependency satisfies the requirements of BCNF as the determinant (LHS: shelter_id) is the primary key which implies it is a super key.
   shelter_id+ = {shelter_id, shelter_name, shelter_loc, pincode, shelter_contact }
   As we can get the information of all attributes from the attribute mentioned on left side of the relation, we can say that this FD satisfies BCNF.

   shelter_id, shelter_name -> address, shelter_contact:
   Proof:
   This functional dependency satisfies the requirements of BCNF as the determinant (LHS: shelter_id, shelter_name) is a super key as we can determine all other attributes using these 2.
   Proof:
   (shelter_id, shelter_name)+ = { shelter_id, shelter_name, shelter_loc, pincode, shelter_contact }
   As we can get the information of all attributes from the attribute mentioned on left side of the relation, we can say that this FD satisfies BCNF.

   shelter_loc -> pin code: This functional dependency does not satisfy the requirements of BCNF as the determinant (LHS: shelter_loc) is not a super key and therefore there is a need of a new table. Shelter location cannot be used to determine each row in the table.

   So, we are splitting this table into 2:
   shelter_branch: (old table)
   shelter_id
   shelter_name
   shelter_loc
   address
   shelter_contact

   shelter_loc_details (new table)
   shelter_loc
   pin_code

   Proof for preservation of Functional Dependencies post-split:
   All the functional dependencies either fall into the old or new table created
   shelter_id -> shelter_name, shelter_loc: This dependency is preserved in the old table
   shelter_id, shelter_name -> address, shelter_contact: This dependency is preserved in the old table
   shelter_loc -> pin_code: This dependency is preserved in the new table

   Thus both tables are in BCNF

   Proof for Lossless Joins:

To ensure that the join is lossless either of the conditions mentioned below should be satisfied:
R1 ∩ R2 -> R1 – R2 or R1 ∩ R2 -> R2 – R1 (where R1 is the old table and R2 is the new table)
R1 ∩ R2 = { shelter_loc }

R1 ∩ R2 -> R1 – R2 = shelter_loc -> {shelter_id, shelter_name, shelter_loc , address, shelter_contact}
R1 ∩ R2 -> R2 – R1 = shelter_loc -> pin_code

Second condition is satisfied and thus this join is said to be lossless.

2. **facilities:**
facilities_id -> f_capacity, f_ani_count, temp_control, outdoor_space, shelter_id:
Proof:
This functional dependency satisfies the requirements of BCNF as the determinant (LHS: facilities_id) is the primary key which implies it is a super key.
facilities_id + = { facilities_id, shelter_id, f_capacity, f_ani_count, f_area, play_areas, temp_control, outdoor_space }
As we can get the information of all attributes from the attribute mentioned on left side of the relation, we can say that this FD satisfies BCNF.

facilities_id, outdoor_space -> f_area, play_areas: This functional dependency satisfies the requirements of BCNF as the determinant (LHS: facilities_id, outdoor_space) is a super key as we can determine all other attributes using these 2.
(facilities_id, outdoor_space) + = { facilities_id, shelter_id, f_capacity, f_ani_count, f_area, play_areas, temp_control, outdoor_space }
As we can get the information of all attributes from the attribute mentioned on left side of the relation, we can say that this FD satisfies BCNF.

This table satisfies conditions of BCNF.

3. **animal_info:**
animal_id -> a_type, a_breed, a_dob, date_of_acquisition, a_vaccine_status: This functional dependency satisfies the requirements of BCNF as the determinant (LHS: animal_id) is the primary key which implies it is a super key.
As we can get the information of all attributes from the attribute mentioned on left side of the relation, we can say that this FD satisfies BCNF.

animal_id, a_type -> animal_features_id: This functional dependency satisfies the requirements of BCNF as the determinant (LHS: animal_id, a_type) is a super key as we can determine all other attributes using these 2.
As we can get the information of all attributes from the attribute mentioned on left side of the relation, we can say that this FD satisfies BCNF.

a_breed -> a_type: This functional dependency does not satisfy the requirements of BCNF as the determinant (LHS: a_breed) is not a super key and therefore there is a need of a new table. Breed of animal cannot be used to determine other attributes in the table.
So, we are splitting this table into 2:
animal_info: (old table)
animal_id

a_breed
a_dob
date_of_acquisition
a_vaccine_status
animal_features_id
animal_breed_info (new table)
a_breed
a_type

Proof for preservation of Functional Dependencies post-split:
All the functional dependencies either fall into the old or new table created
animal_id -> a_type, a_breed, a_dob, date_of_acquisition, a_vaccine_status: This dependency is preserved in the old table
animal_id, a_type -> animal_features_id: This dependency is preserved in the old table
a_breed -> a_type: This dependency is preserved in the new table

Thus, both tables are in BCNF

Proof for Lossless Joins:
To ensure that the join is lossless either of the conditions mentioned below should be satisfied:
R1 ∩ R2 -> R1 – R2 or R1 ∩ R2 -> R2 – R1 (where R1 is the old table and R2 is the new table)
R1 ∩ R2 = { a_breed }

R1 ∩ R2 -> R1 – R2 = a_breed -> { animal_id, a_dob, date_of_acquisition, a_vaccine_status, animal_features_id}
R1 ∩ R2 -> R2 – R1 = a_breed -> a_type

Second condition is satisfied and thus this join is said to be lossless.

4. **animal_features:**
animal_features_id -> a_color, a_weight, a_height, a_condition, a_eye_color: This functional dependency satisfies the requirements of BCNF as the determinant (LHS: animal_features_id) is primary key which implies it is a super key.
{ animal_features_id+ = {animal_features_id, a_color, a_weight, a_height, a_eye_color, a_condition}
As we can get the information of all attributes from the attribute mentioned on left side of the relation, we can say that this FD satisfies BCNF.

animal_features_id, a_condition -> a_color, a_weight, a_height: This functional dependency satisfies the requirements of BCNF as the determinant (LHS: animal_features_id, a_condition) is a super key as animal_feature_id and a_condition can be used to get the other details in other attributes.
(animal_features_id, a_condition)+ ={ animal_features_id, a_condition, a_color, a_weight, a_height, a_eye_color}
As we can get the information of all attributes from the attribute mentioned on left side of the relation, we can say that this FD satisfies BCNF.

5. **adoption_details:**

ad_ID -> ad_name, ad_contact, ad_date, ad_fee, payment_ID : This functional dependency satisfies the requirements of BCNF as the determinant (LHS: ad_ID) is the primary key which implies it is a super key.
ad_ID+ = { ad_ID, payment_ID, ad_name, ad_contact, ad_date, ad_fee}
As we can get the information of all attributes from the attribute mentioned on left side of the relation, we can say that this FD satisfies BCNF.

ad_ID, ad_date -> payment_ID, ad_name, ad_contact: This functional dependency satisfies the requirements of BCNF as the determinant (LHS: ad_ID, ad_date) is a super key as ad_ID and ad_can be used to get the other details in other attributes.
ad_ID, ad_date+ ={ ad_ID, ad_date payment_ID, ad_name, ad_contact, ad_fee}
As we can get the information of all attributes from the attribute mentioned on left side of the relation, we can say that this FD satisfies BCNF.

6. **surrender_details:**
s_id -> s_date, s_name: This functional dependency satisfies the requirements of BCNF as the determinant (LHS: s_id) is the primary key which implies it is a super key.
s_id+ ={s_id, s_date, s_name, s_reason}
As we can get the information of all attributes from the attribute mentioned on left side of the relation, we can say that this FD satisfies BCNF.

s_id, s_date -> s_reason: This functional dependency satisfies the requirements of BCNF as the determinant (LHS: s_id, s_date) is a super key as s_id and s_date can be used to get the other details in other attributes.
s_id, s_date+ ={ s_id, s_date, s_name, s_reason}
As we can get the information of all attributes from the attribute mentioned on left side of the relation, we can say that this FD satisfies BCNF.

7. **nutrition:**
food_id -> food_type: This functional dependency satisfies the requirements of BCNF as the determinant (LHS: food_id) is the primary key which implies it is a super key.
food_id+ ={ food_id, food_type, food_quantity}
As we can get the information of all attributes from the attribute mentioned on left side of the relation, we can say that this FD satisfies BCNF.

food_id, food_type -> food_quantity: This functional dependency satisfies the requirements of BCNF as the determinant (LHS: food_id, food_type) is a super key as food_id and food_type can be used to get the other details in other attributes.
(food_id, food_type)+ ={ food_id, food_type, food_quantity}
As we can get the information of all attributes from the attribute mentioned on left side of the relation, we can say that this FD satisfies BCNF.

8. **training_history:**
training_ID -> trainer_name, training_date, training_type: This functional dependency satisfies the requirements of BCNF as the determinant (LHS: training_ID) is the primary key which implies it is a super key.

training_ID+ ={ training_ID, trainer_name, training_date, training_type, training_stage}
As we can get the information of all attributes from the attribute mentioned on left side of the relation, we can say that this FD satisfies BCNF.

training_ID, training_date -> training_stage: This functional dependency satisfies the requirements of BCNF as the determinant (LHS: training_ID, training_date) is a super key as training_ID and training_date can be used to get the other details in other attributes.
training_ID, training_date+ ={ training_ID, training_date, trainer_name, training_type, training_stage}
As we can get the information of all attributes from the attribute mentioned on left side of the relation, we can say that this FD satisfies BCNF.

9. **grooming_history:**
grooming_id -> groomer_name, grooming_date: This functional dependency satisfies the requirements of BCNF as the determinant (LHS: grooming_id) is the primary key which implies it is a super key.
grooming_id+ ={ grooming_id, groomer_name, grooming_date, grooming_type}
As we can get the information of all attributes from the attribute mentioned on left side of the relation, we can say that this FD satisfies BCNF.

grooming_id, groomer_name -> grooming_type: This functional dependency satisfies the requirements of BCNF as the determinant (LHS: grooming_id, groomer_name) is a super key as grooming_id and groomer_name can be used to get the other details in other attributes.
grooming_id, groomer_name+={ grooming_id, groomer_name, grooming_date, grooming_type}
As we can get the information of all attributes from the attribute mentioned on left side of the relation, we can say that this FD satisfies BCNF.

10. **medical_history:**
medical_id -> disease, vaccination_id: This functional dependency satisfies the requirements of BCNF as the determinant (LHS: medical_id) is the primary key which implies it is a super key.
As we can get the information of all attributes from the attribute mentioned on left side of the relation, we can say that this FD satisfies BCNF.

medical_id, disease -> vaccination_id, treatment_date: This functional dependency satisfies the requirements of BCNF as the determinant (LHS: medical_id, disease) is a super key as we can determine all other attributes using these on LHS.
As we can get the information of all attributes from the attribute mentioned on left side of the relation, we can say that this FD satisfies BCNF.

disease -> vaccination_id: This functional dependency does not satisfy the requirements of BCNF as the determinant (LHS: disease) is not a super key and therefore there is a need of a new table. Disease name cannot be used to determine each row in the table.
So, we are splitting this table into 2:
medical_history: (old table)
medical_id
disease

treatment_date
disease_mitigation (new table)
disease
vaccination_id

Proof for preservation of Functional Dependencies post-split:
All the functional dependencies either fall into the old or new table created
medical_id -> disease, vaccination_id: : This dependency is preserved in the old table
medical_id, disease -> vaccination_id, treatment_date: : This dependency is preserved in the old table
disease -> vaccination_id: : This dependency is preserved in the new table

Thus, both tables are in BCNF

Proof for Lossless Joins:
To ensure that the join is lossless either of the conditions mentioned below should be satisfied:
R1 ∩ R2 -> R1 – R2 or R1 ∩ R2 -> R2 – R1 (where R1 is the old table and R2 is the new table)
R1 ∩ R2 = { a_breed }

R1 ∩ R2 -> R1 – R2 = disease -> { medical_id, treatment_date}
R1 ∩ R2 -> R2 – R1 = disease -> vaccination_id

Second condition is satisfied and thus this join is said to be lossless.

11. **vaccination:**
vaccination_id -> vaccination_name, vaccination_date: This functional dependency satisfies the requirements of BCNF as the determinant (LHS: vaccination_id) is the primary key which implies it is a super key.
vaccination_id+ = {vaccination_id, vaccination_name, vaccination_date, vaccination_dosage }
As we can get the information of all attributes from the attribute mentioned on left side of the relation, we can say that this FD satisfies BCNF.

vaccination_id, vaccination_name -> vaccination_dosage: This functional dependency satisfies the requirements of BCNF as the determinant (LHS: vaccination_id, vaccination_name) is a super key as vaccination_id and vaccination_name can be used to get the other details in other attributes.
vaccination_id, vaccination_name + ={ vaccination_id, vaccination_name, vaccination_date, vaccination_dosage}
As we can get the information of all attributes from the attribute mentioned on left side of the relation, we can say that this FD satisfies BCNF.

12. **employees_staff_details:**
emp_id -> emp_name, emp_contact, emp_training, emp_education, emp_criminal_record: This functional dependency satisfies the requirements of BCNF as the determinant (LHS: emp_id) is the primary key which implies it is a super key.
As we can get the information of all attributes from the attribute mentioned on left side of the relation, we can say that this FD satisfies BCNF.

emp_id, emp_contact -> emp_criminal_record: This functional dependency satisfies the requirements of BCNF as the determinant (LHS: emp_id, emp_contact) is a super key as we can determine all other attributes using these on LHS.
As we can get the information of all attributes from the attribute mentioned on left side of the relation, we can say that this FD satisfies BCNF.

emp_education -> emp_level: This functional dependency does not satisfy the requirements of BCNF as the determinant (LHS: emp_education) is not a super key and therefore there is a need of a new table. emp_education name cannot be used to determine each row in the table.
emp_education -> emp_department: This functional dependency does not satisfy the requirements of BCNF as the determinant (LHS: emp_education) is not a super key and therefore there is a need of a new table. emp_education name cannot be used to determine each row in the table.

So, we are splitting this table into 2:
employees_staff_details*: (old table)*
emp_id
emp_name
emp_contact
emp_training
emp_education
emp_criminal_record
employees_stage *(new table)*
emp_education
emp_department
emp_level

Proof for preservation of Functional Dependencies post-split:
All the functional dependencies either fall into the old or new table created
emp_id -> emp_name, emp_contact, emp_training, emp_education, emp_criminal_record: This dependency is preserved in the old table
emp_id, emp_contact -> emp_criminal_record: This dependency is preserved in the old table
emp_education -> emp_level: This dependency is preserved in the new table
emp_education -> emp_department: This dependency is preserved in the new table

Thus, both tables are in BCNF

Proof for Lossless Joins:
To ensure that the join is lossless either of the conditions mentioned below should be satisfied:
R1 ∩ R2 -> R1 – R2 or R1 ∩ R2 -> R2 – R1 (where R1 is the old table and R2 is the new table)
R1 ∩ R2 = { emp_education }

R1 ∩ R2 -> R1 – R2 = emp_education -> { emp_id, emp_name, emp_contact, emp_training , emp_criminal_record}
R1 ∩ R2 -> R2 – R1 = emp_education -> emp_department, emp_level

Second condition is satisfied and thus this join is said to be lossless with the combination of third and fourth FDs.

13. **sponsor:**
sponsor_id -> sponsor_name, sponsor_contact: This functional dependency satisfies the requirements of BCNF as the determinant (LHS: sponsor_id) is the primary key which implies it is a super key.

As we can get the information of all attributes from the attribute mentioned on left side of the relation, we can say that this FD satisfies BCNF.

sponsor_id, sponsor_name, sponsor_contact -> sponsorship_type: This functional dependency satisfies the requirements of BCNF as the determinant (LHS: sponsor_id, sponsor_name, sponsor_contact) is a super key as sponsor_id, sponsor_name and sponsor_contact can be used to get the other details in other attributes.
As we can get the information of all attributes from the attribute mentioned on left side of the relation, we can say that this FD satisfies BCNF.

14. **payment_details:**
payment_ID -> sponsor_id, payment_Date, payment_type: This functional dependency satisfies the requirements of BCNF as the determinant (LHS: payment_ID) is the primary key which implies it is a super key.
As we can get the information of all attributes from the attribute mentioned on left side of the relation, we can say that this FD satisfies BCNF.

payment_ID, sponsor_id -> amount, payment_Date, payment_type: This functional dependency satisfies the requirements of BCNF as the determinant (LHS: payment_ID, sponsor_id) is a super key as payment_ID and sponsor_id can be used to get the other details in other attributes.
As we can get the information of all attributes from the attribute mentioned on left side of the relation, we can say that this FD satisfies BCNF.

15. **veterinary:**
vet_id -> vet_name, vet_contact, education, work_exp: This functional dependency satisfies the requirements of BCNF as the determinant (LHS: vet_id) is the primary key which implies it is a super key.
As we can get the information of all attributes from the attribute mentioned on left side of the relation, we can say that this FD satisfies BCNF.

vet_name, vet_contact -> work_exp: This functional dependency does not satisfy the requirements of BCNF as the determinant (LHS: vet_name, vet_contact) is not a super key and therefore there is a need of a new table. vet_name and vet_contact cannot be used to determine each row in the table.

So, we are splitting this table into 2:
veterinary*: (old table)*

vet_id
vet_name
vet_contact
education
vet_edu *(new table)*
vet_name
vet_contact
work_exp

Proof for preservation of Functional Dependencies post-split:
All the functional dependencies either fall into the old or new table created
emp_education -> emp_department: This dependency is preserved in the new table
vet_id -> vet_name, vet_contact, education, work_exp: This dependency is preserved in the old table
vet_name, vet_contact -> work_exp: This dependency is preserved in the new table

Thus, both tables are in BCNF

Proof for Lossless Joins:
To ensure that the join is lossless either of the conditions mentioned below should be satisfied:
R1 ∩ R2 -> R1 – R2 or R1 ∩ R2 -> R2 – R1 (where R1 is the old table and R2 is the new table)
R1 ∩ R2 = { vet_name, vet_contact }

R1 ∩ R2 -> R1 – R2 = vet_name, vet_contact -> { vet_id, education}
R1 ∩ R2 -> R2 – R1 = vet_name, vet_contact -> work_exp

Second condition is satisfied and thus this join is said to be lossless with the combination of third and fourth FDs.

16. **animal_transfer_records:**
transfer_id -> transfer_date, animal_id: This functional dependency satisfies the requirements of BCNF as the determinant (LHS: transfer_id) is the primary key which implies it is a super key. As we can get the information of all attributes from the attribute mentioned on left side of the relation, we can say that this FD satisfies BCNF.

transfer_id, animal_id -> transfer_date, transfer_from, transfer_to: This functional dependency satisfies the requirements of BCNF as the determinant (LHS: transfer_id, animal_id) is a super key as transfer_id, animal_id can be used to get the other details in other attributes. As we can get the information of all attributes from the attribute mentioned on left side of the relation, we can say that this FD satisfies BCNF.

17. **employee_payroll:**
swipe_id -> swipe_date, check_in_time, check_out_time, emp_id: This functional dependency satisfies the requirements of BCNF as the determinant (LHS: swipe_id) is the primary key which implies it is a super key.

As we can get the information of all attributes from the attribute mentioned on left side of the relation, we can say that this FD satisfies BCNF.

emp_id -> hourly_pay: This functional dependency does not satisfy the requirements of BCNF as the determinant (LHS: emp_id) is not a super key and therefore there is a need of a new table. emp_id name cannot be used to determine each row in the table as there might be multiple swipes for each employee id.
So, we are splitting this table into 3:
employee_payroll*: (old table)*
swipe_id
swipe_date
check_in_time
check_out_time
emp_id
employee_pay*(new table)*
emp_id
hourly_pay

Proof for preservation of Functional Dependencies post-split:
All the functional dependencies either fall into the old or new table created
swipe_id -> swipe_date, check_in_time, check_out_time, emp_id: This dependency is preserved in the old table
emp_id -> hourly_pay: This dependency is preserved in the new table

Thus, both tables are in BCNF

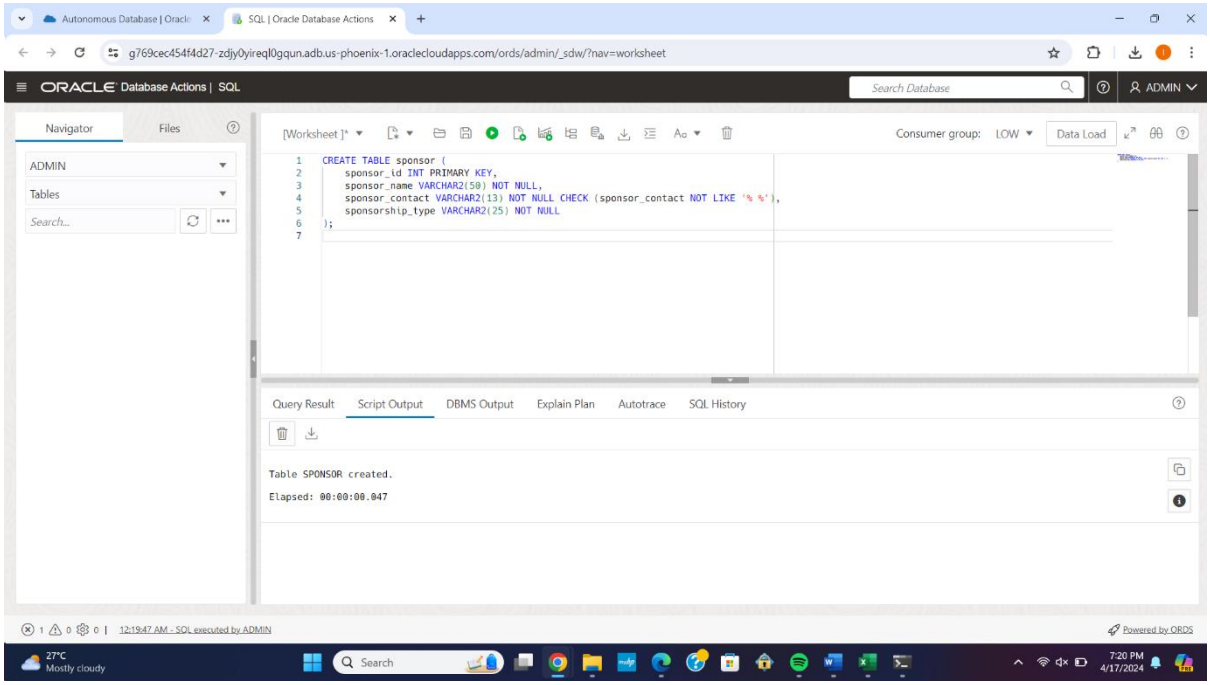
Proof for Lossless Joins:
To ensure that the join is lossless either of the conditions mentioned below should be satisfied:
R1 ∩ R2 -> R1 – R2 or R1 ∩ R2 -> R2 – R1 (where R1 is the old table and R2 is the new table)
R1 ∩ R2 = { emp_id }

R1 ∩ R2 -> R1 – R2 = emp_id -> { swipe_id, swipe_date, check_in_time, check_out_time }
R1 ∩ R2 -> R2 – R1 = emp_id -> hourly_pay

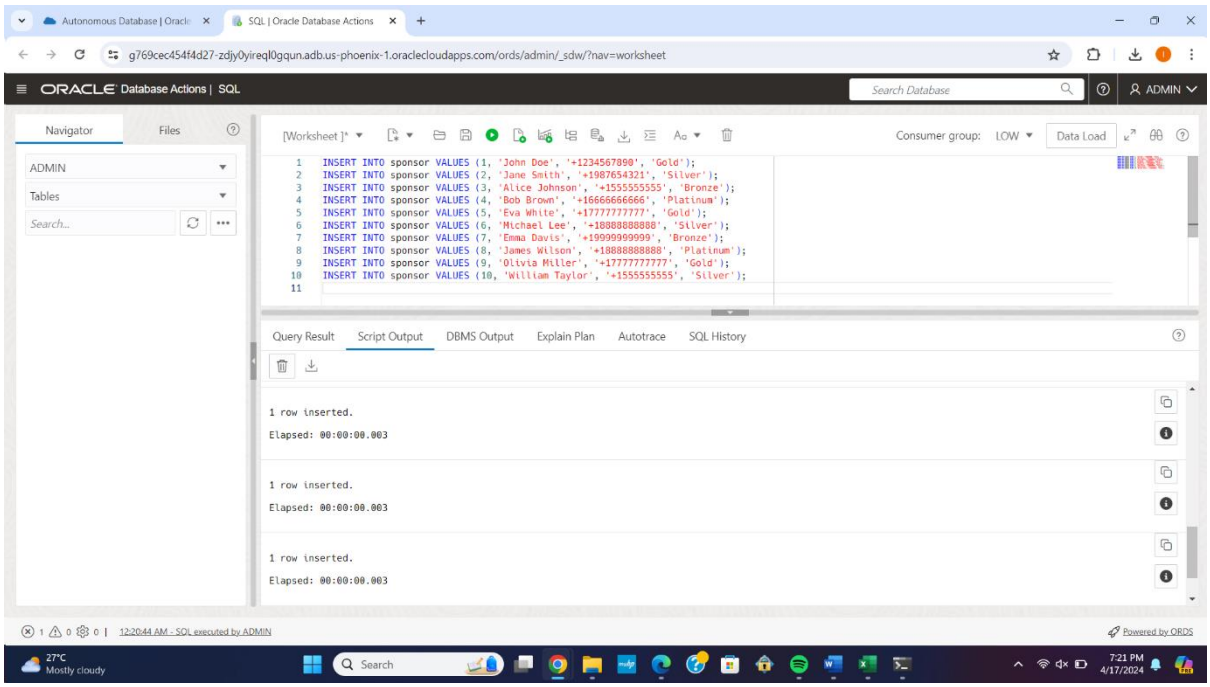
Second condition is satisfied and thus this join is said to be lossless with the combination of third and fourth FDs.

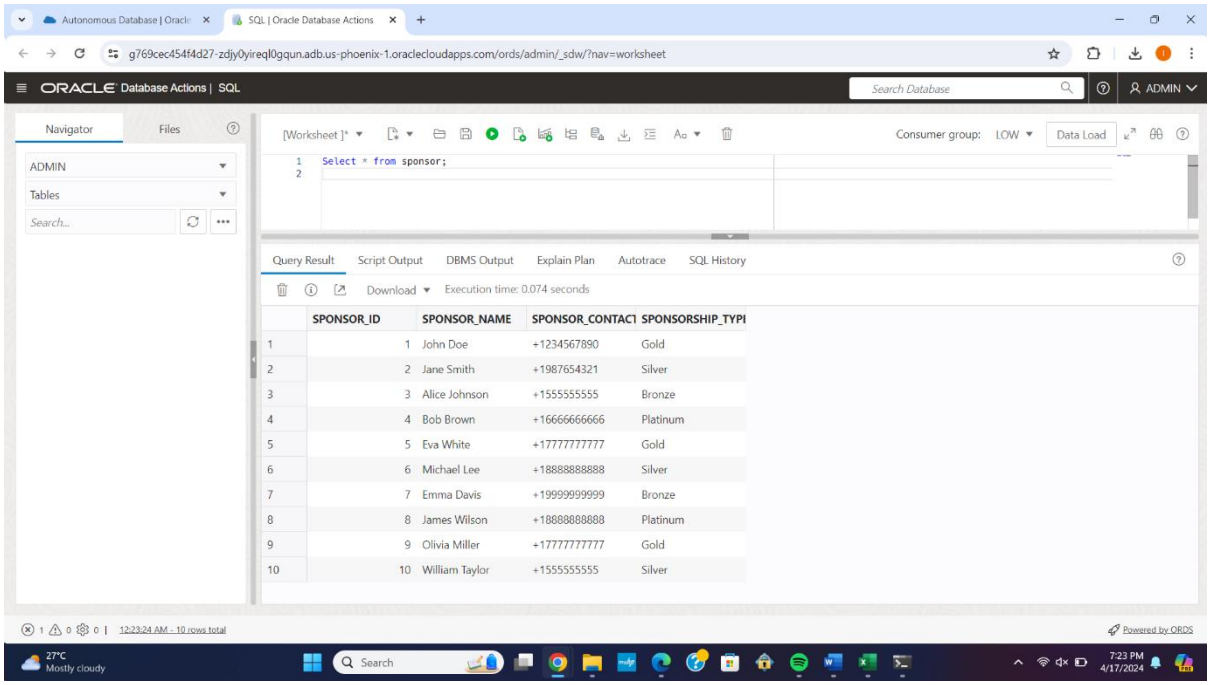
Updated Database Screenshots:

Creating a table named sponsor

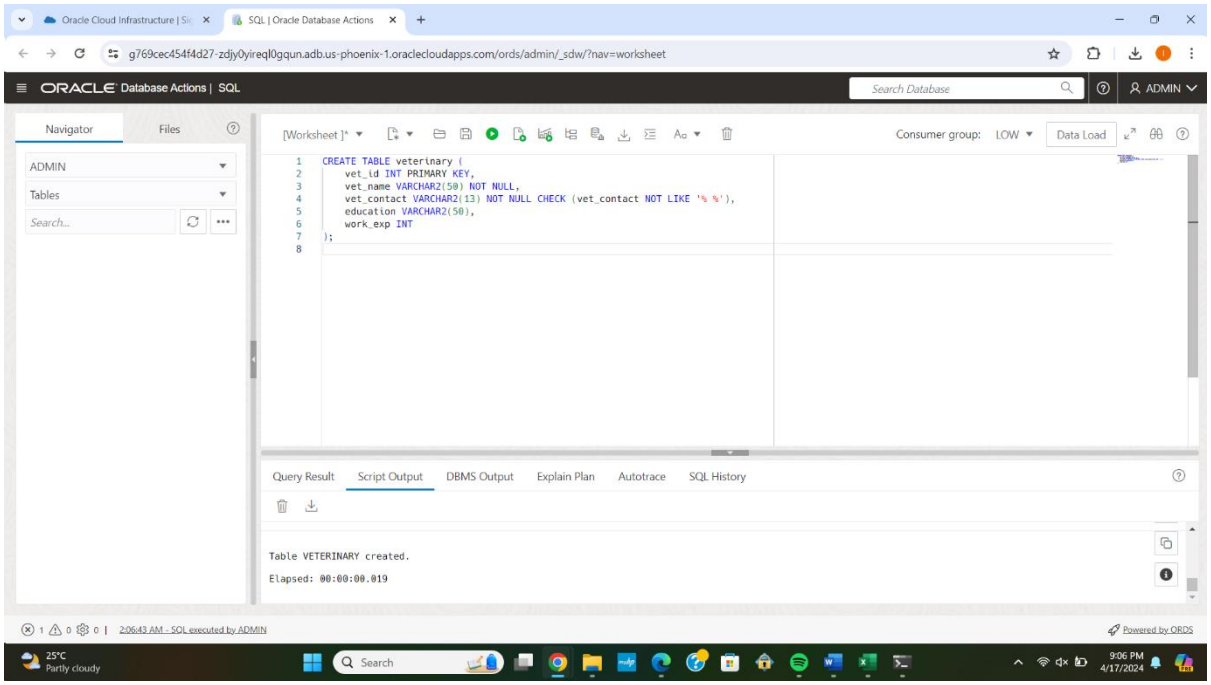


Inserting values to sponsor

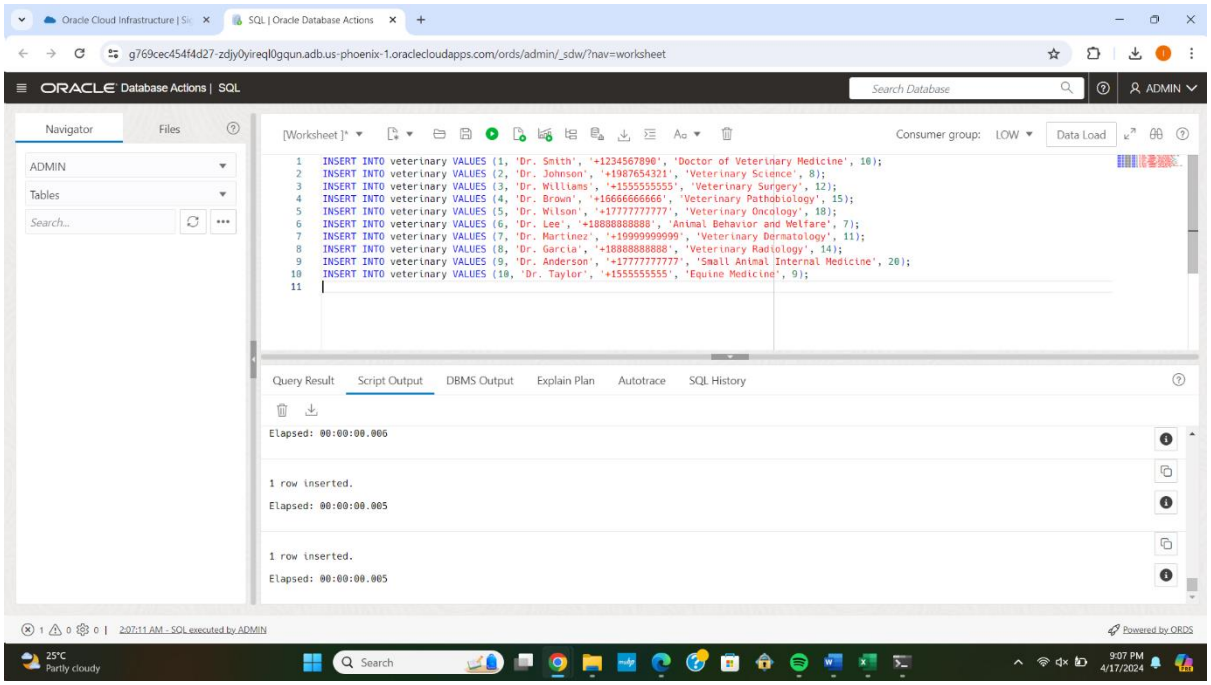




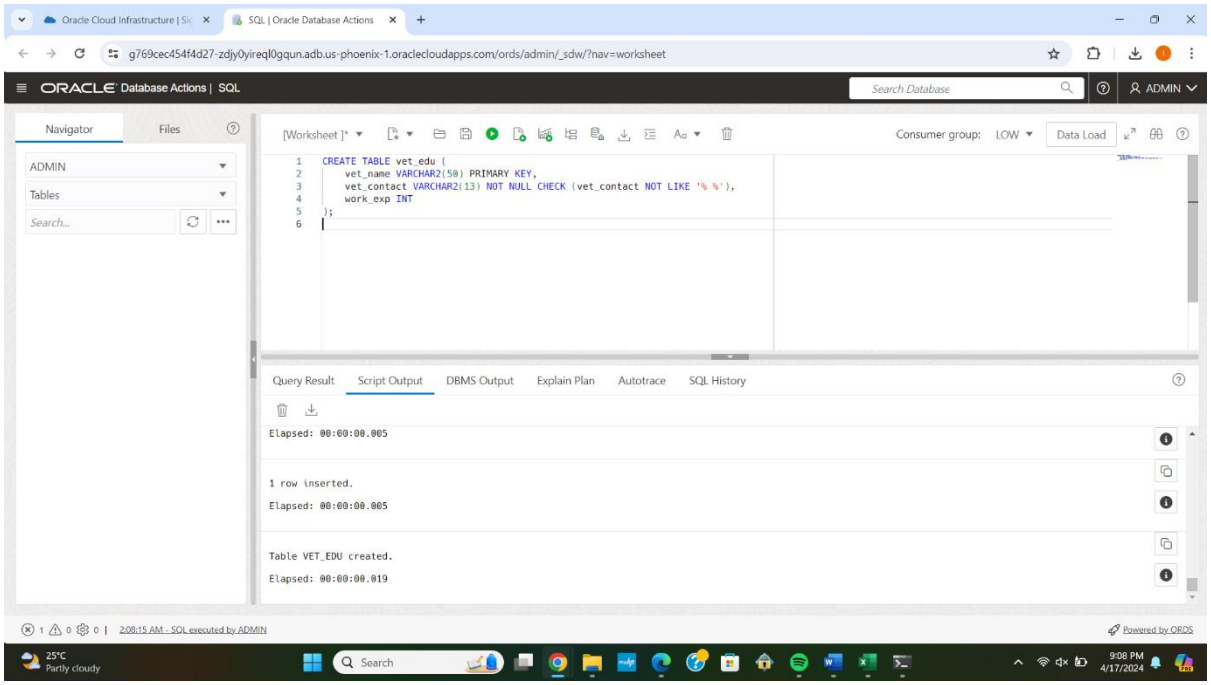
Creating a table named veterinary



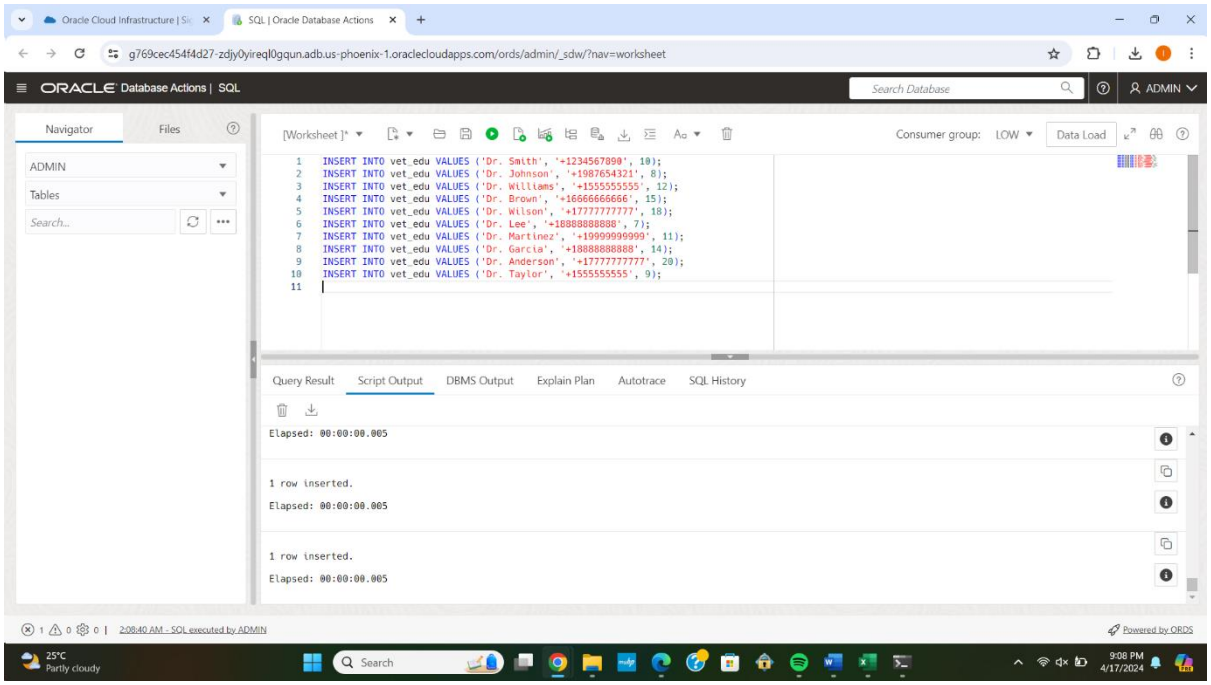
Inserting values to veterinary

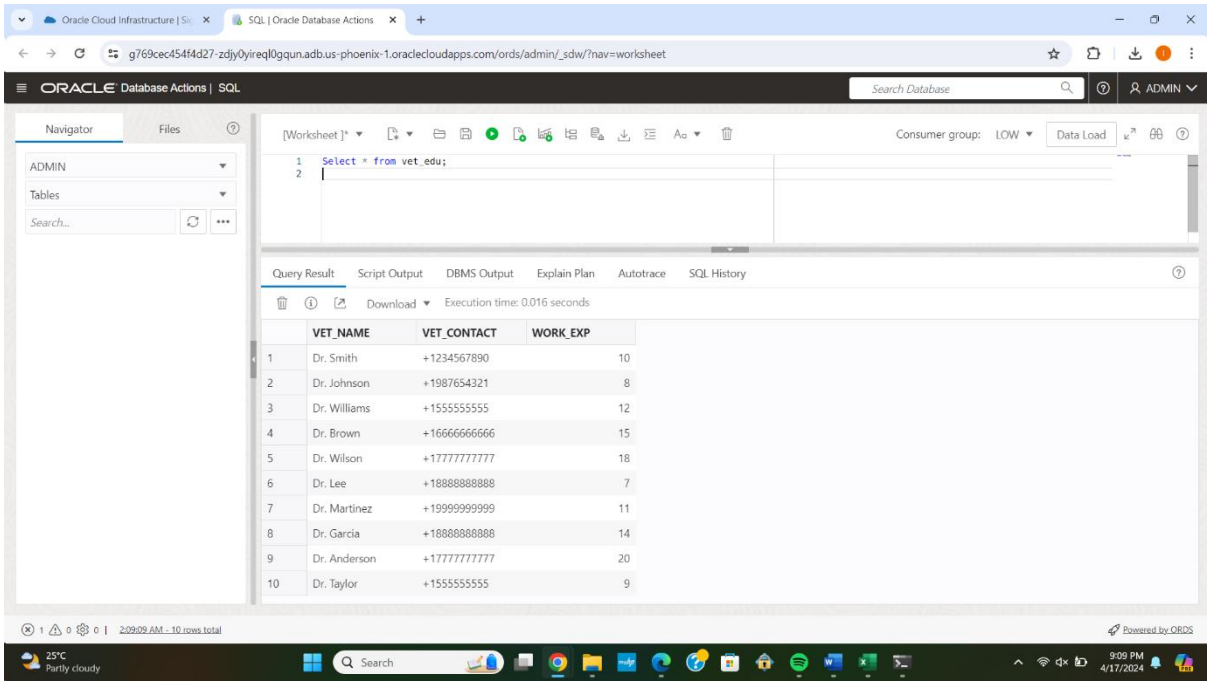


Creating a table named vet_edu(New)

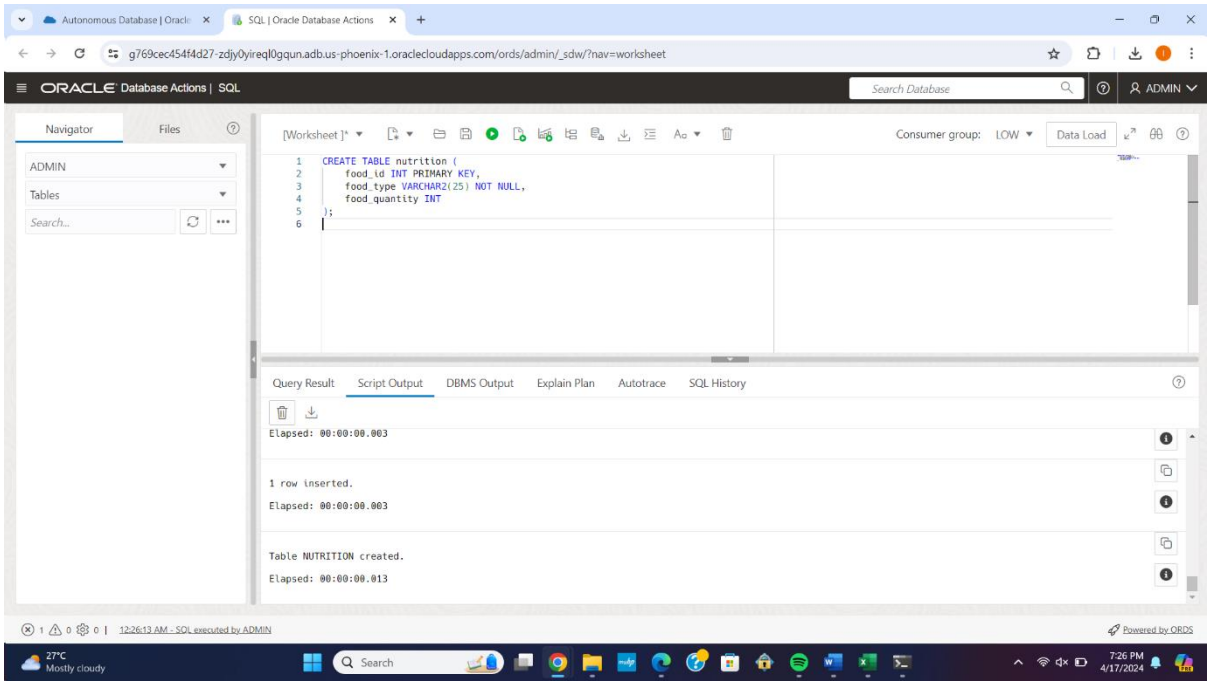


Inserting values to vet_edu(New)

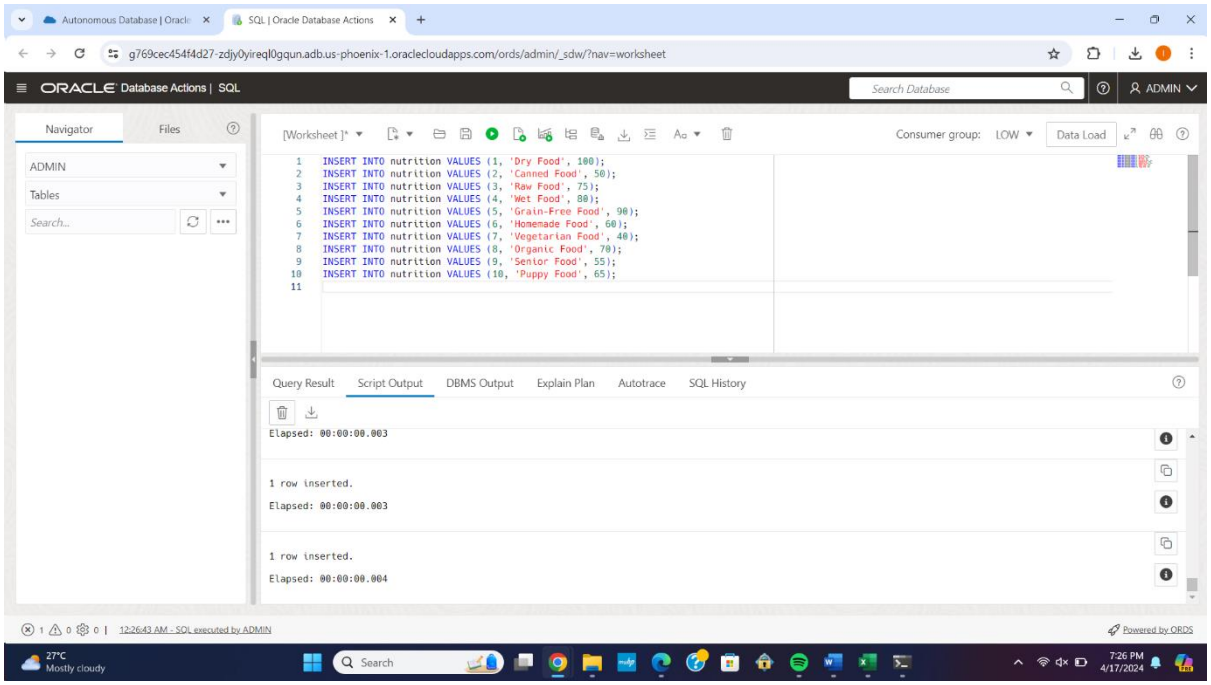


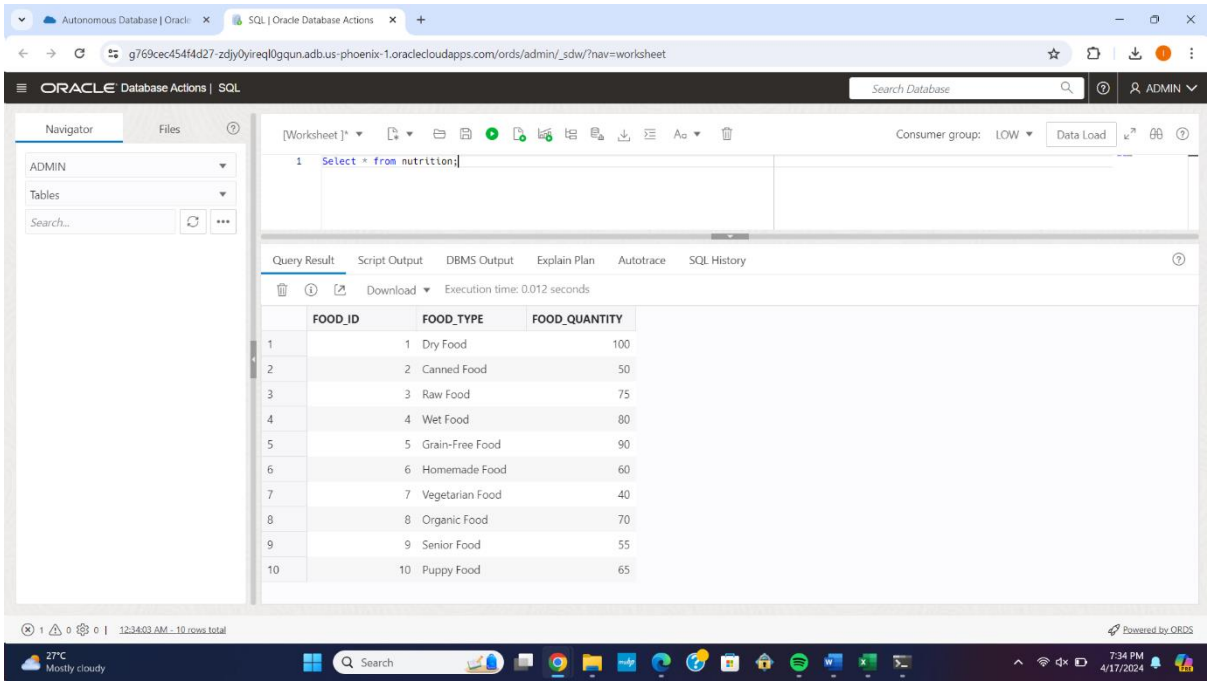


Creating a table named nutrition

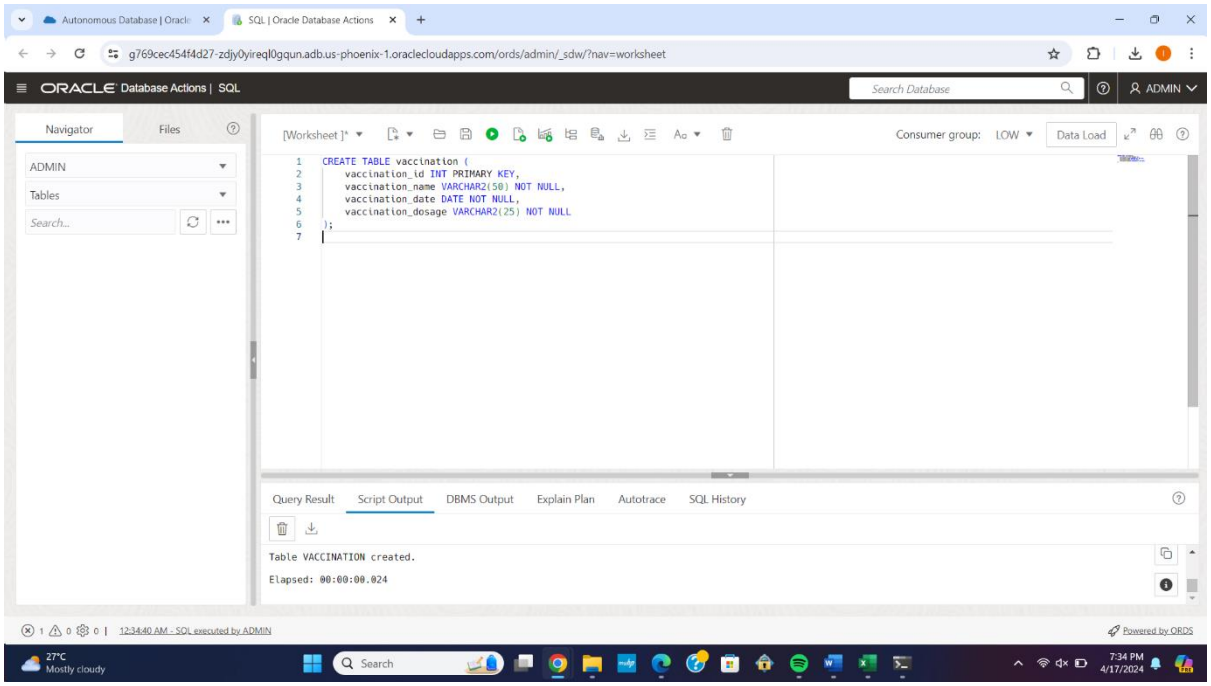


Inserting values to nutrition

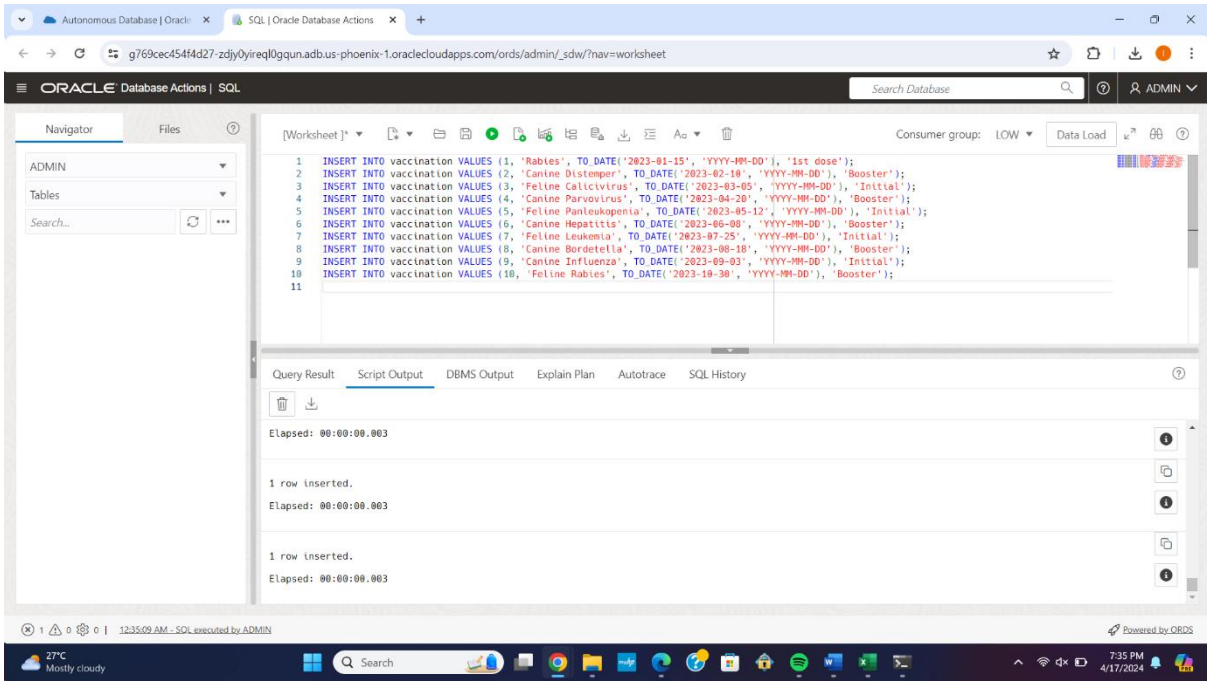


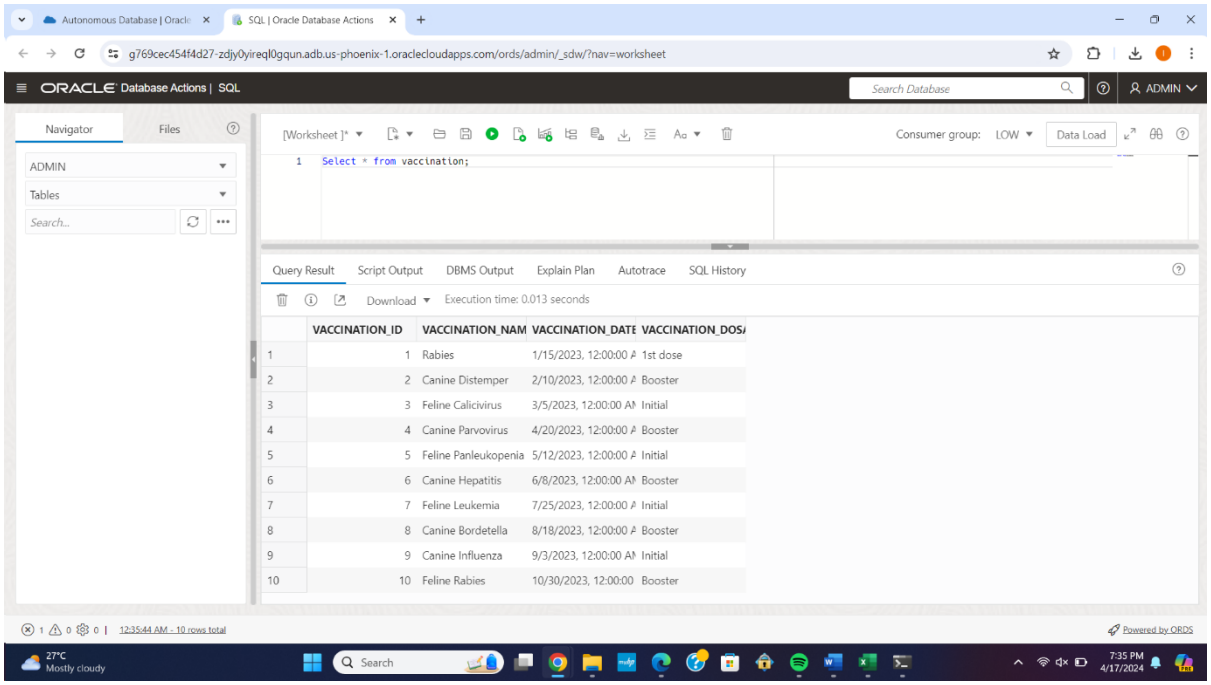


Creating a table named vaccination

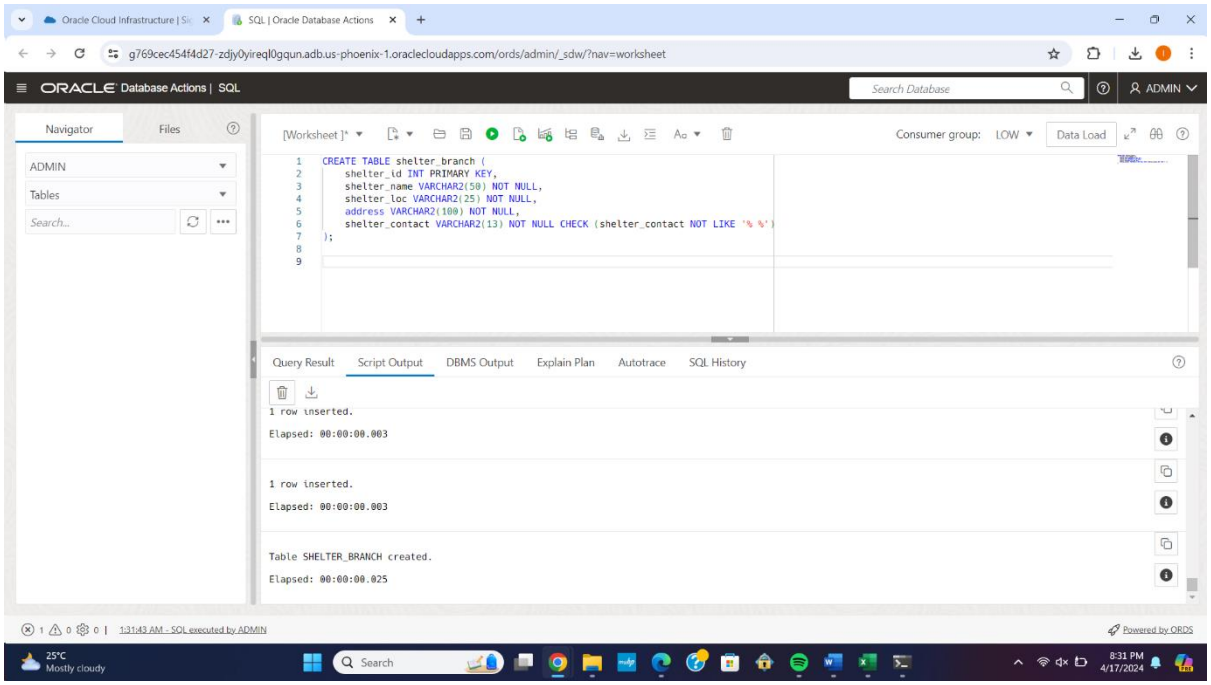


Inserting values to vaccination

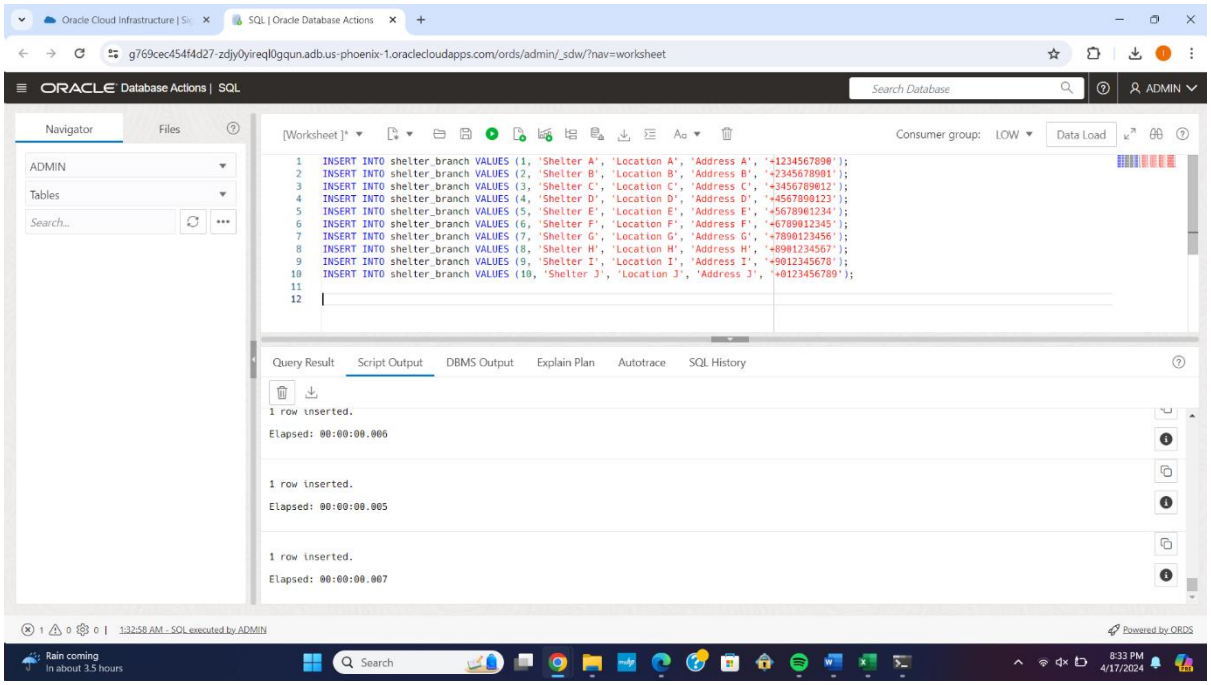


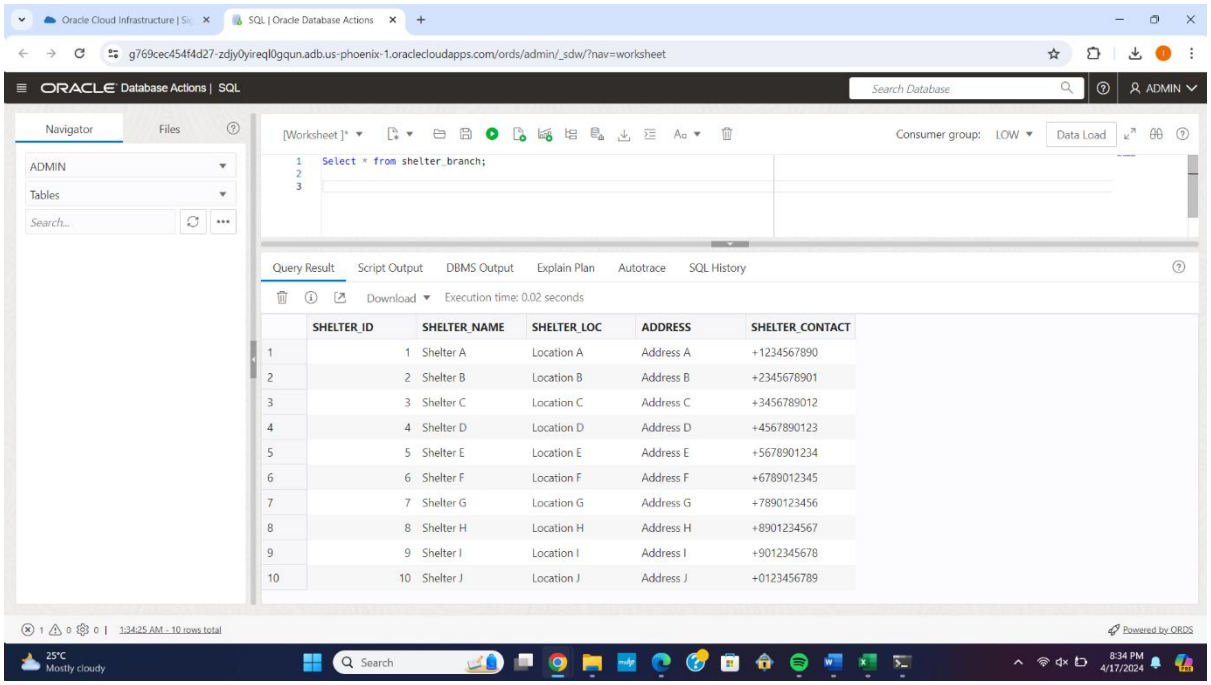


Creating a table named shelter_branch

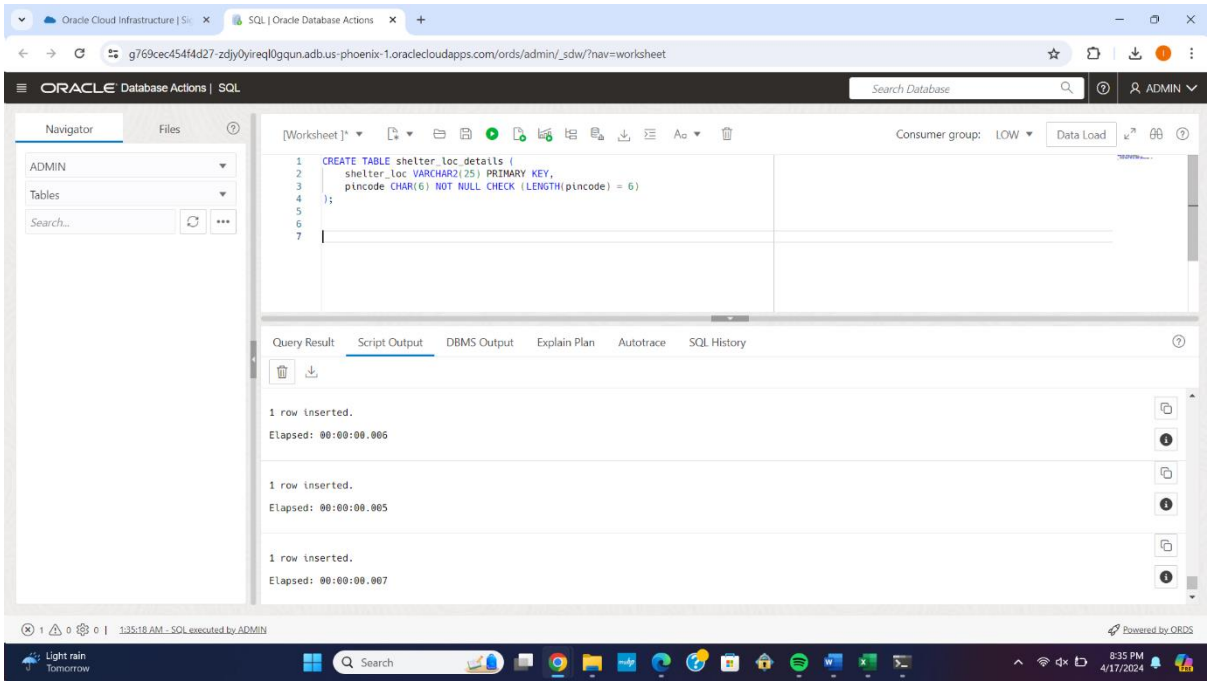


Inserting values to shelter_branch

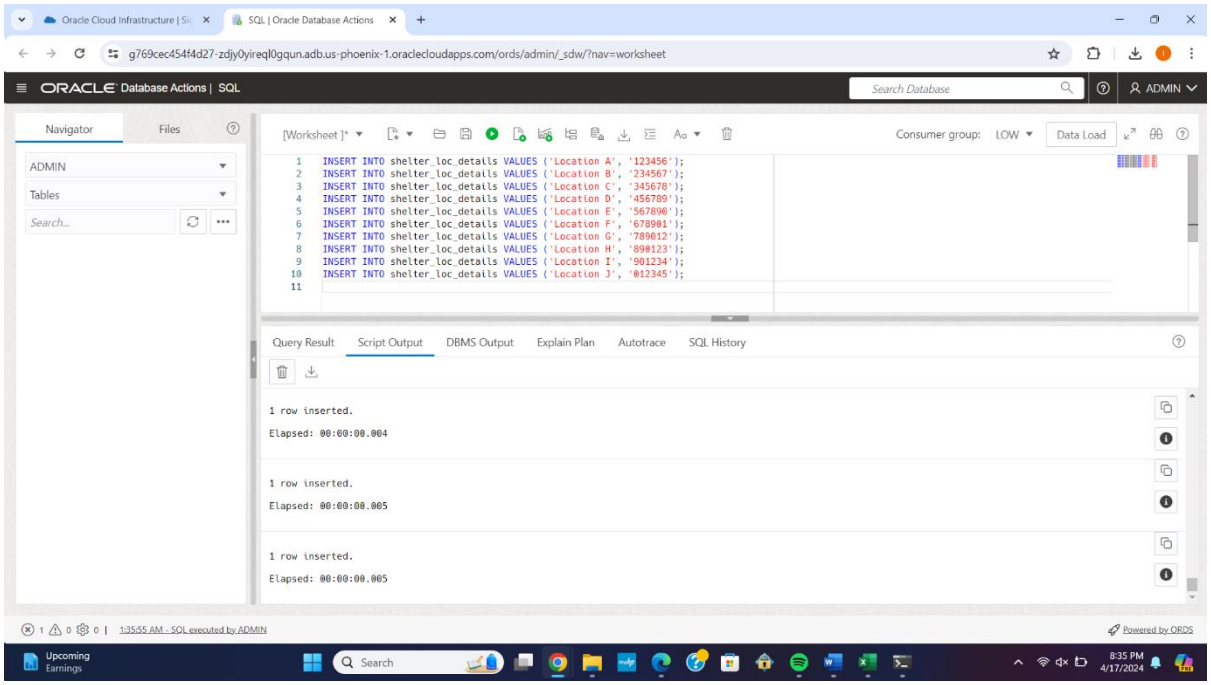


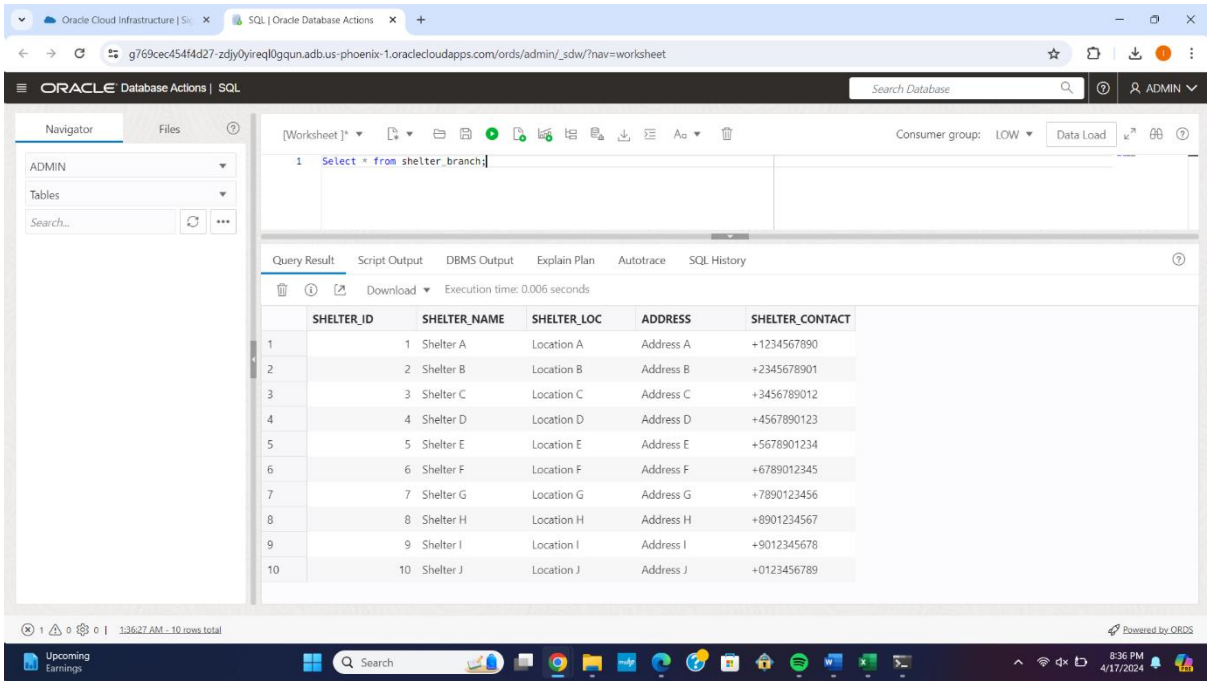


Creating a table named shelter_loc_details(New)

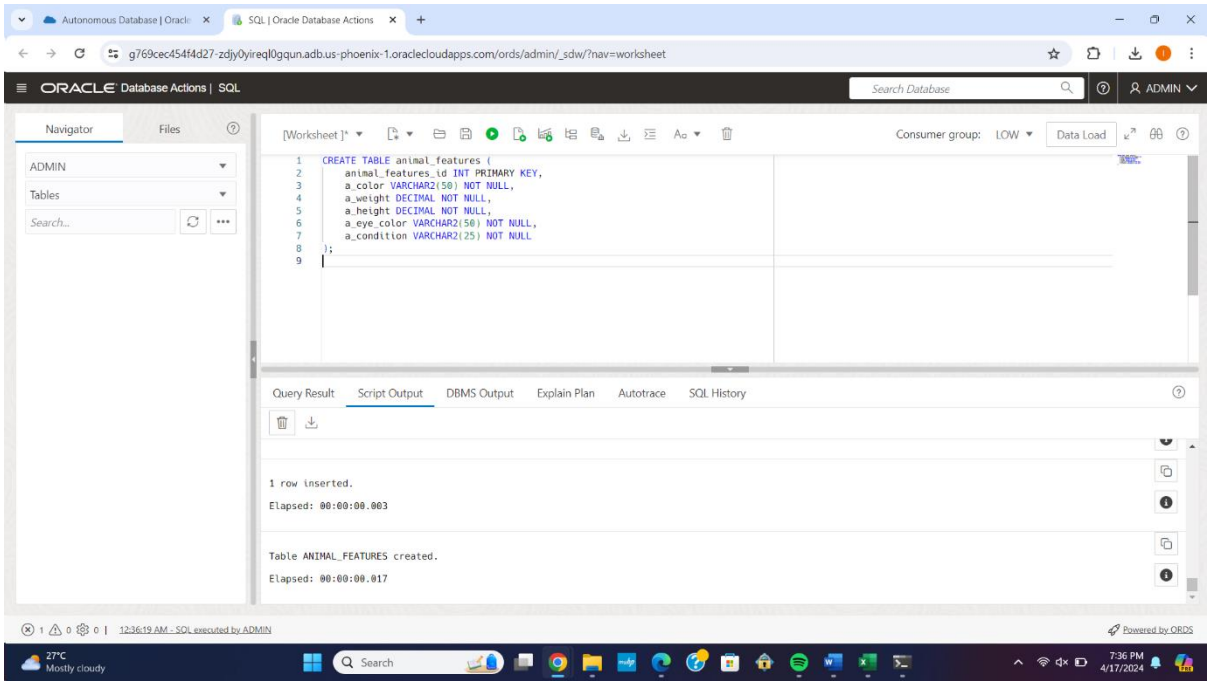


Inserting values to shelter_loc_details(New)

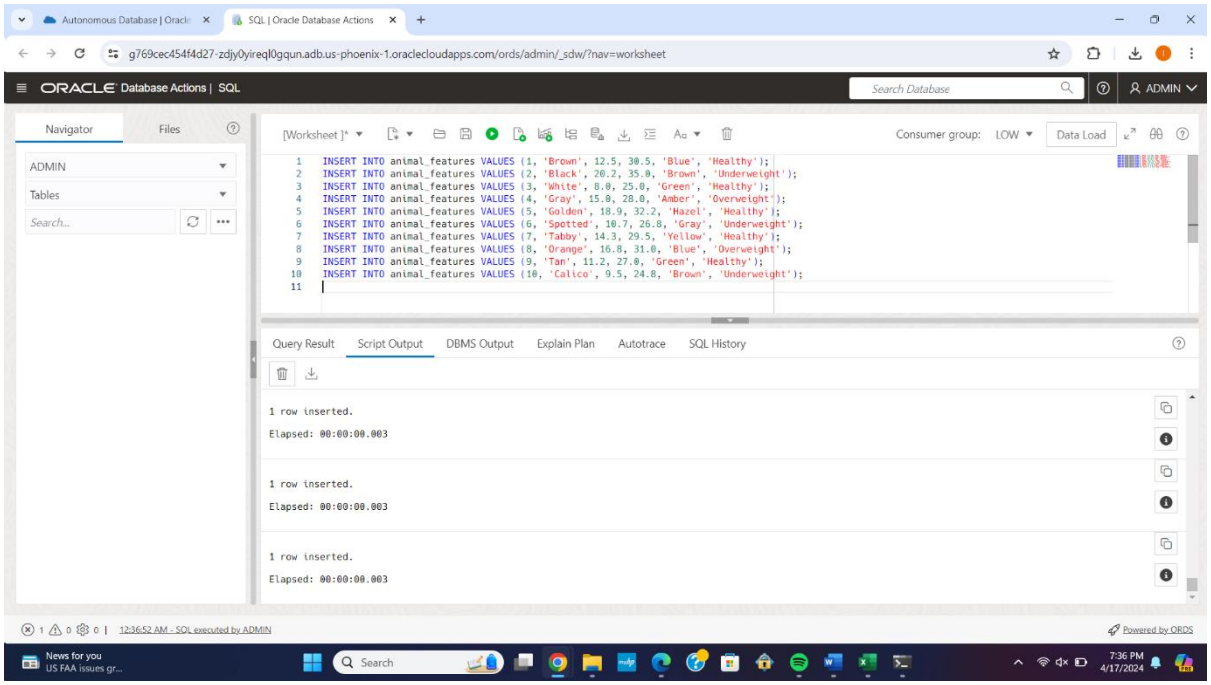


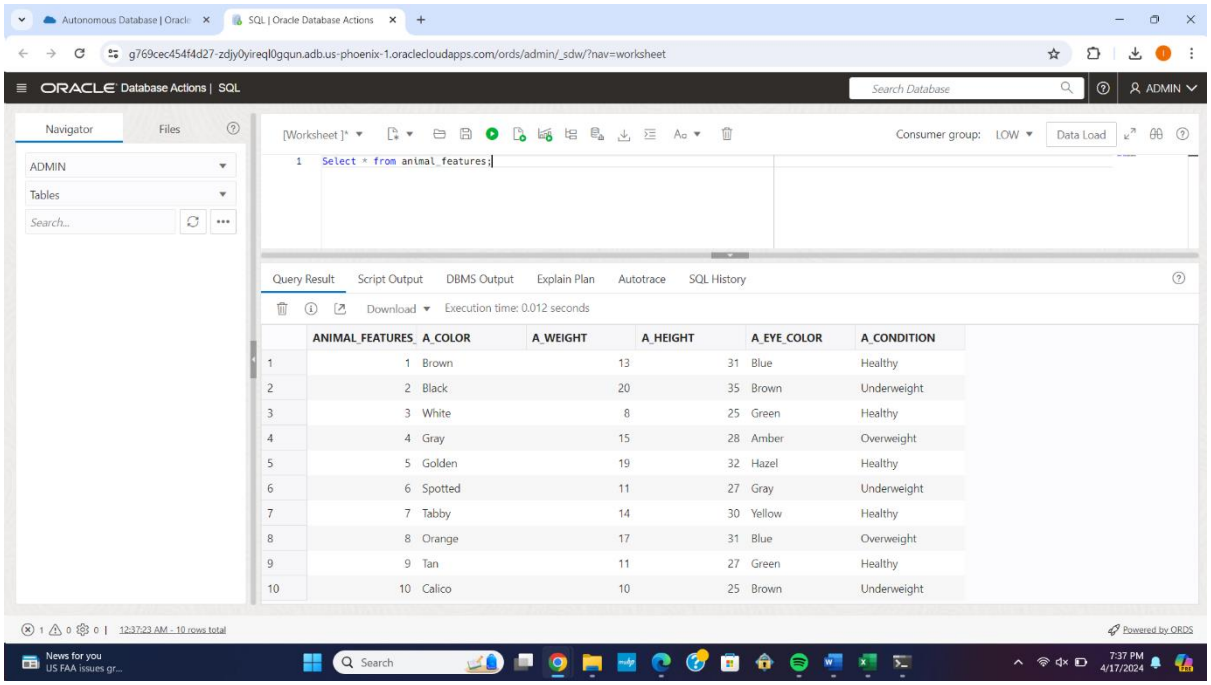


Creating a table named animal_features

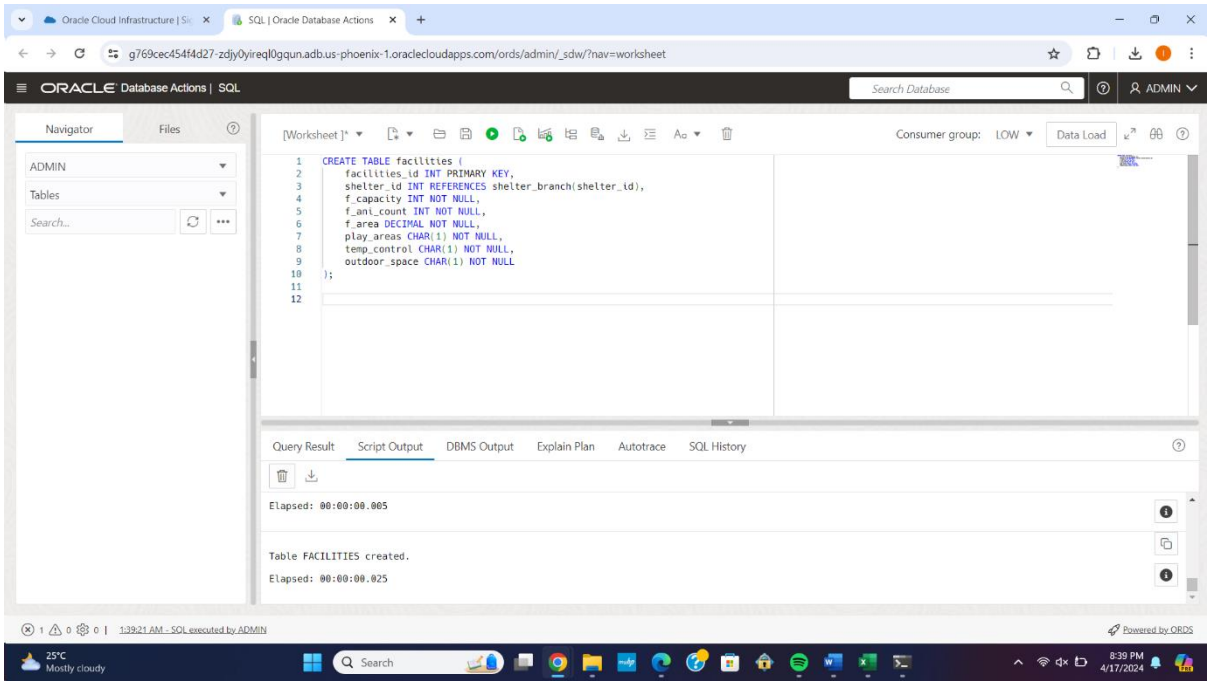


Inserting values to animal_features

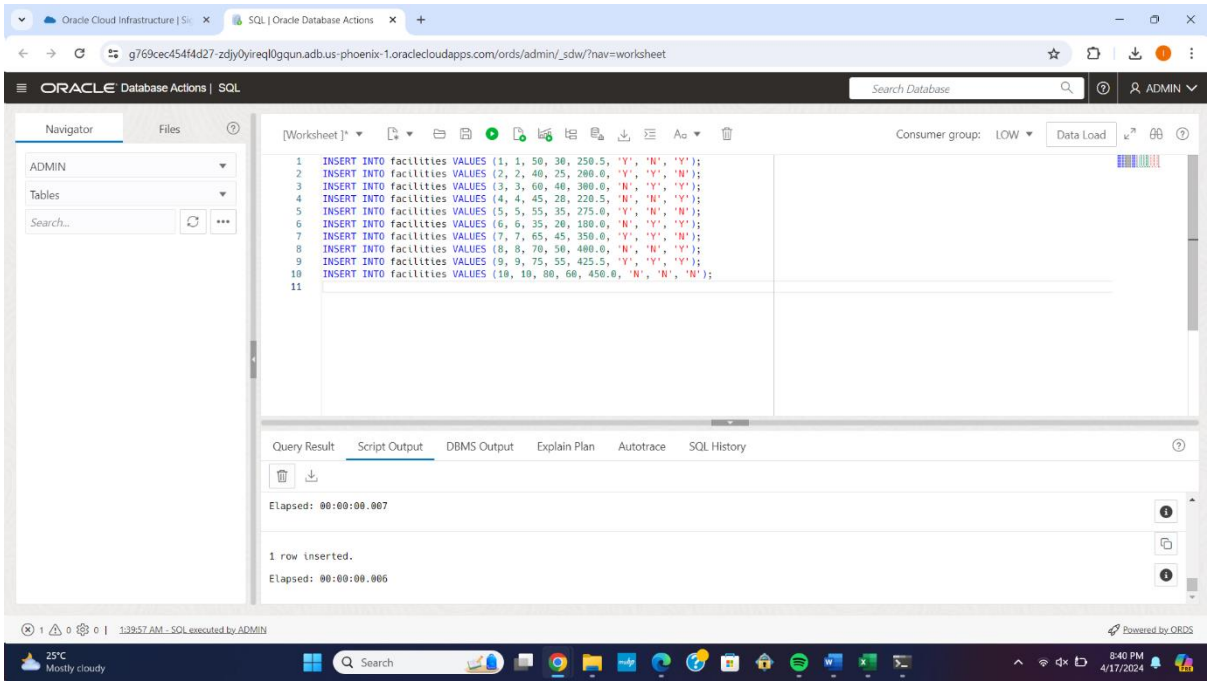




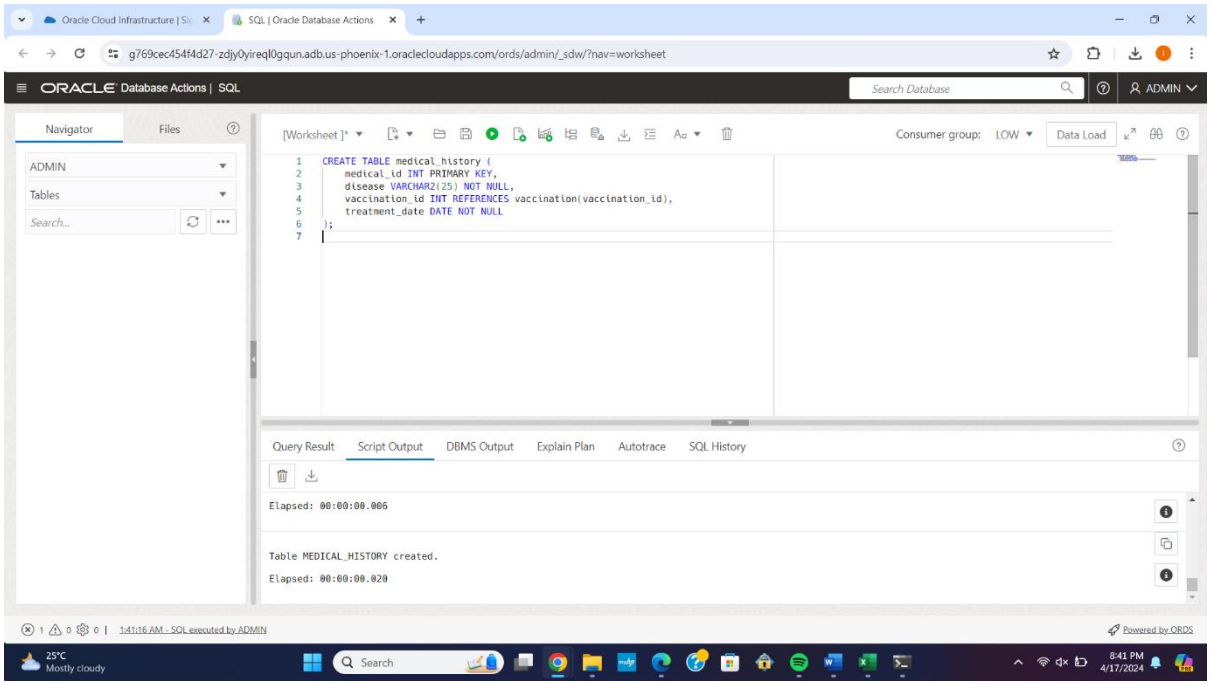
Creating a table named facilities



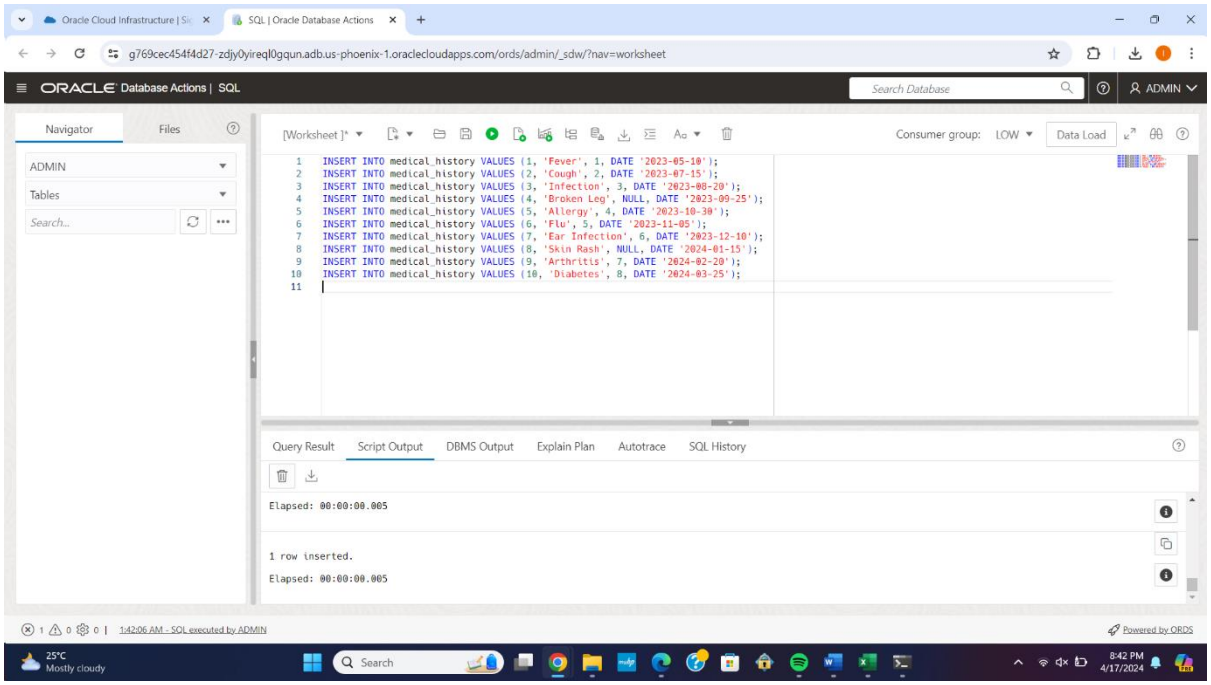
Inserting values to facilities

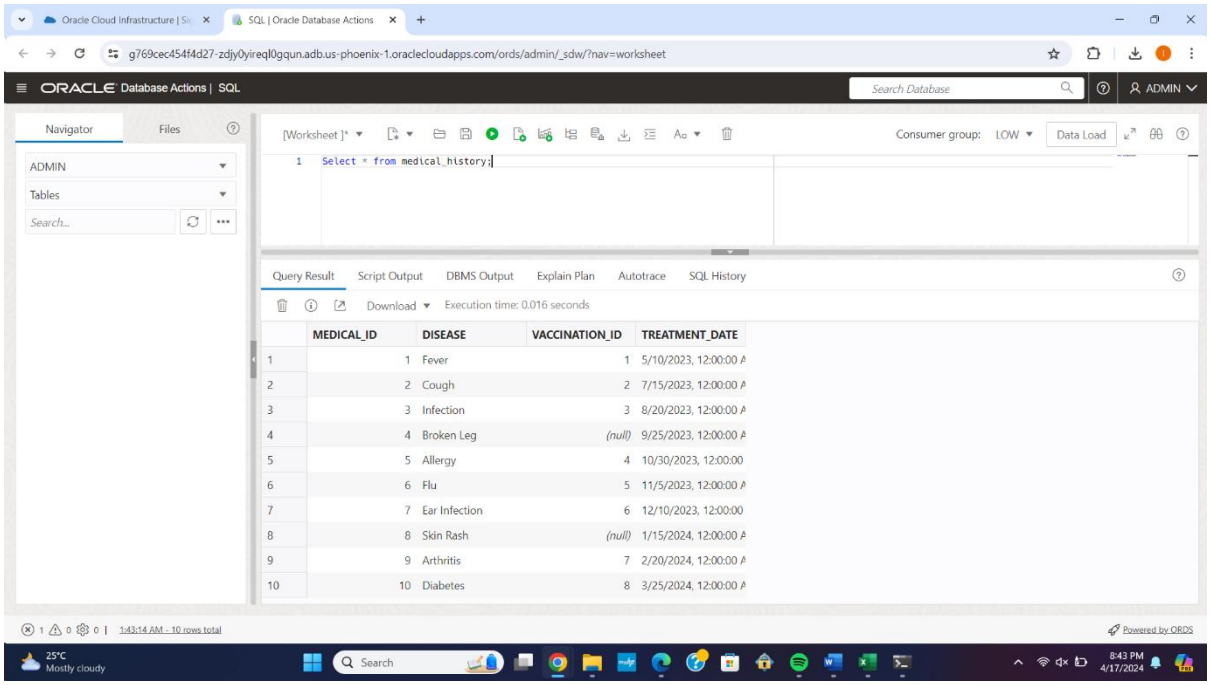


Creating a table named medical_history

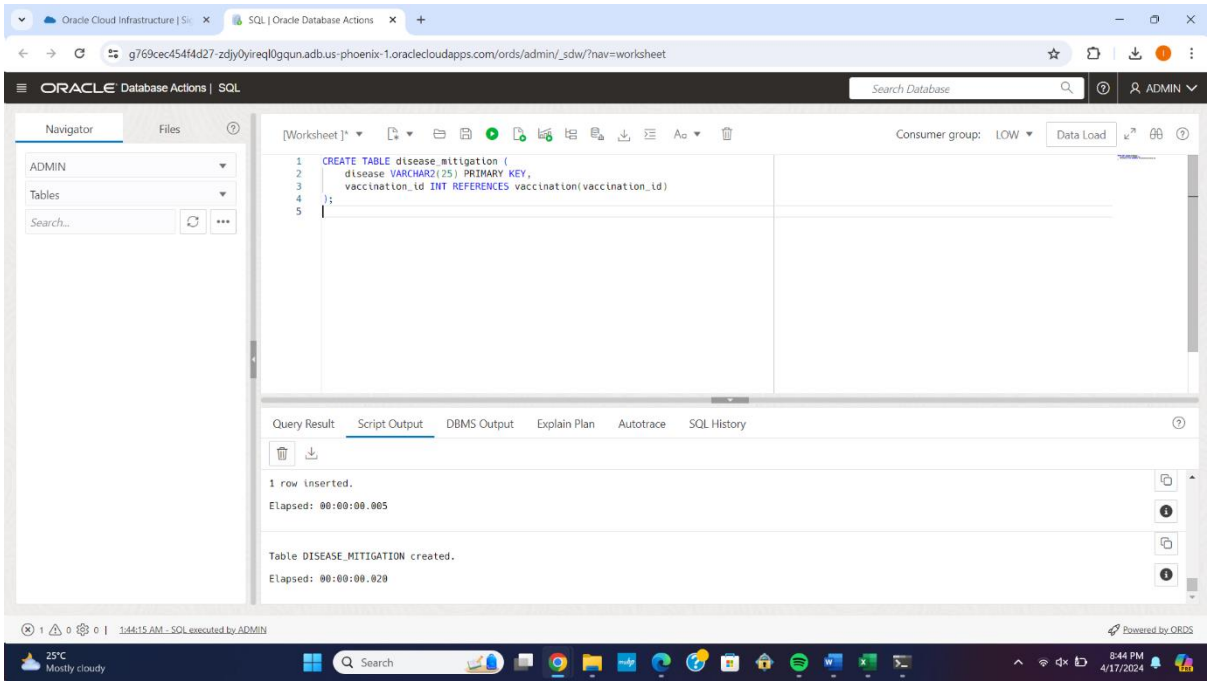


Inserting values to medical_history

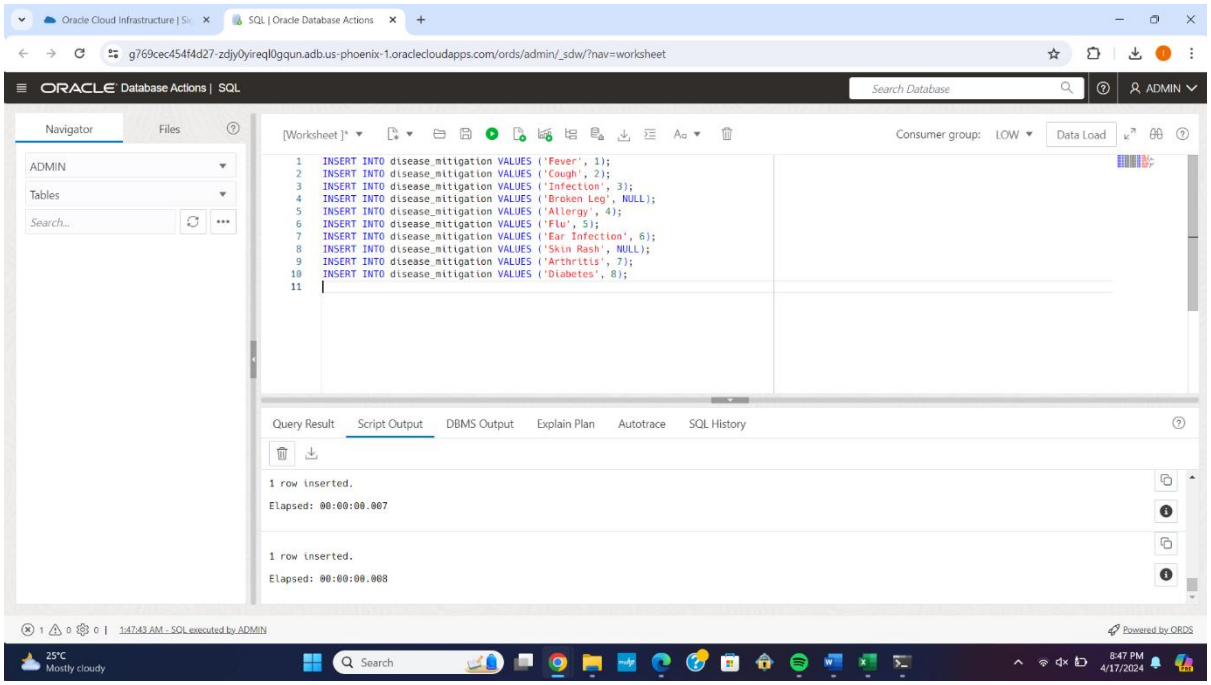


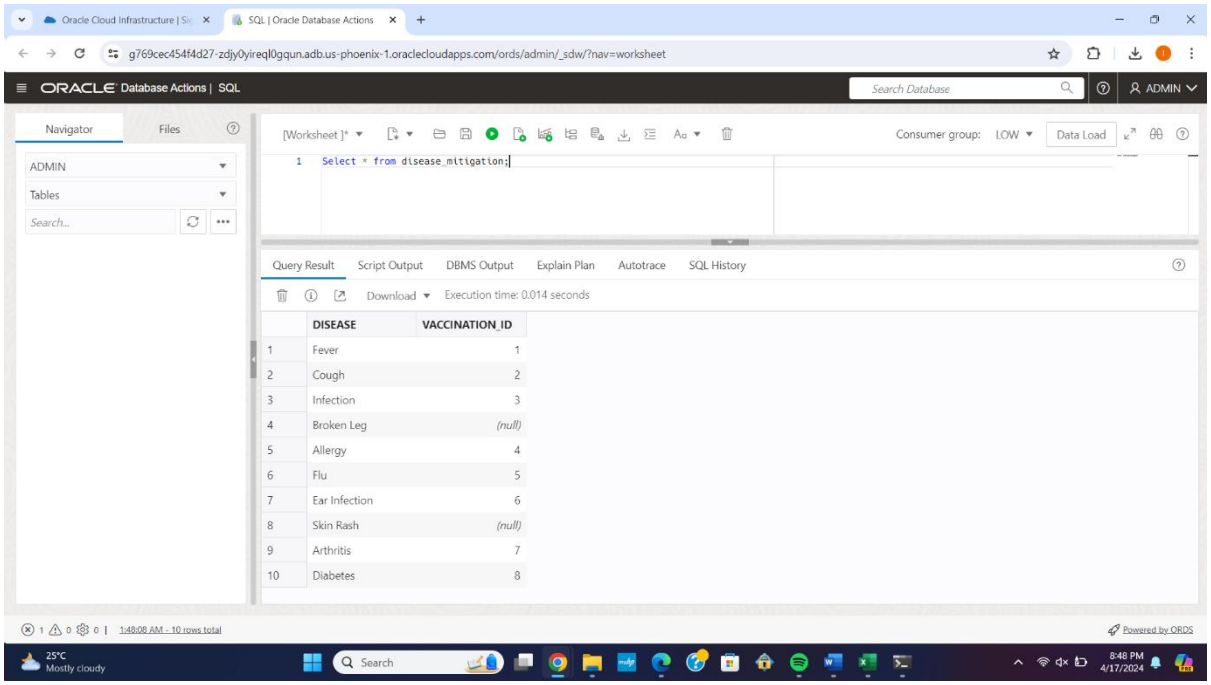


Creating a table named disease_mitigation (New)

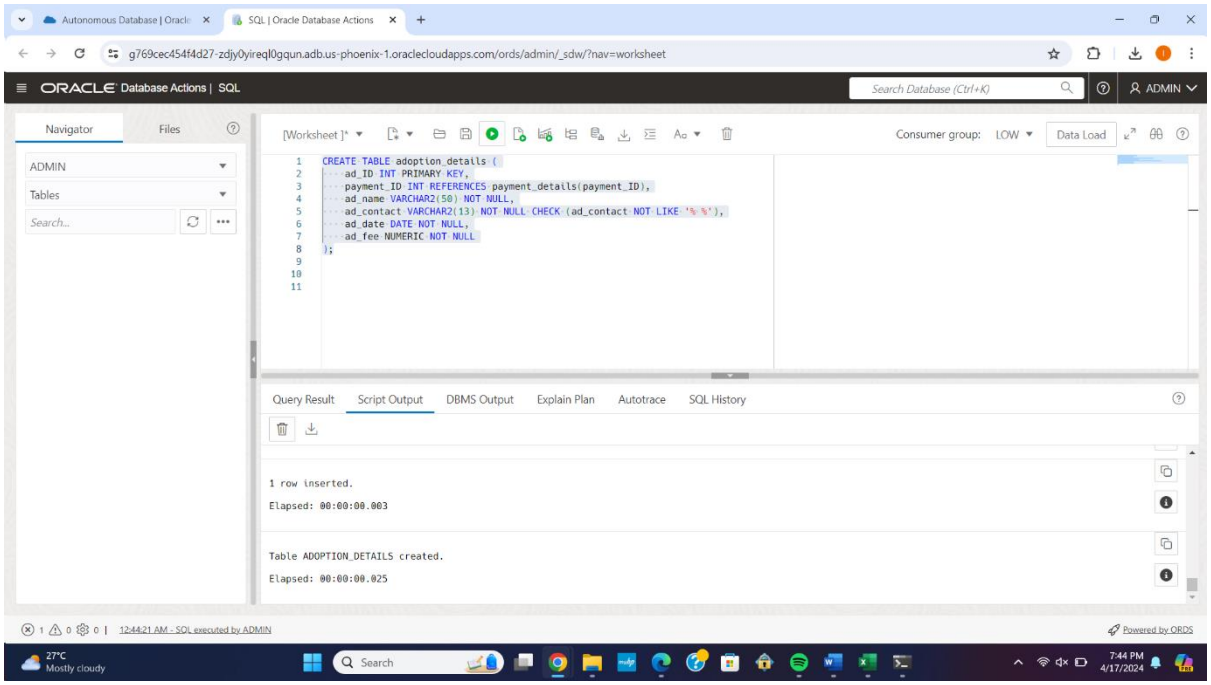


Inserting values to disease_mitigation (New)

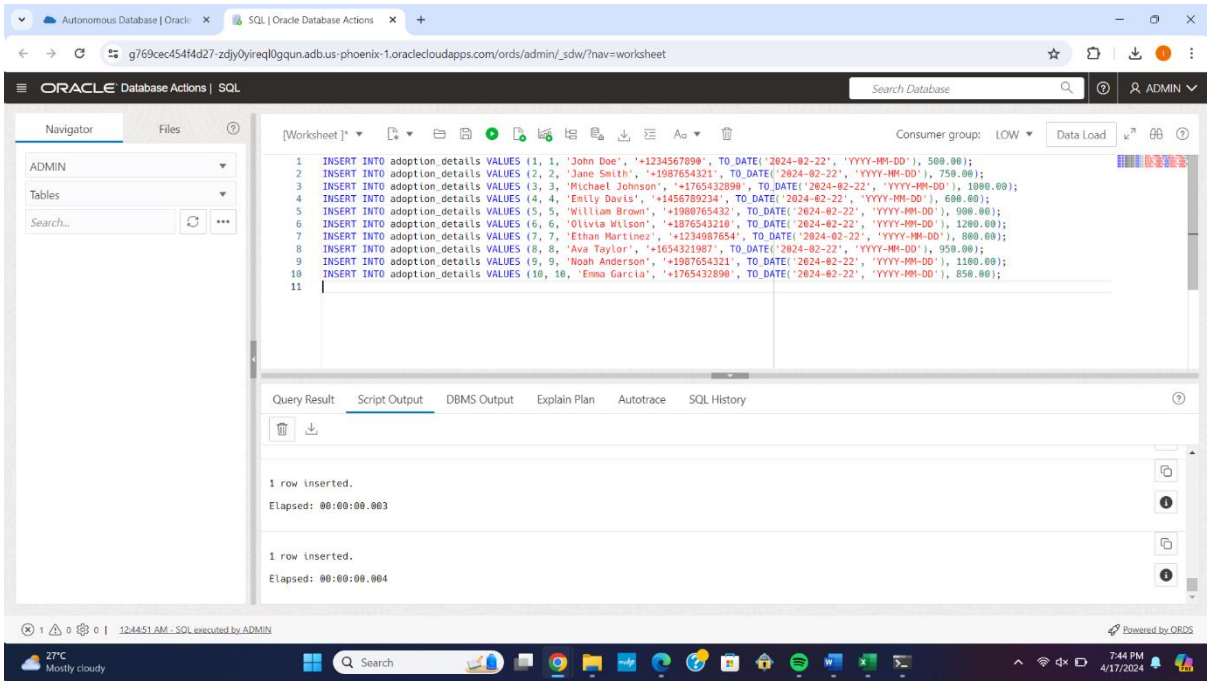


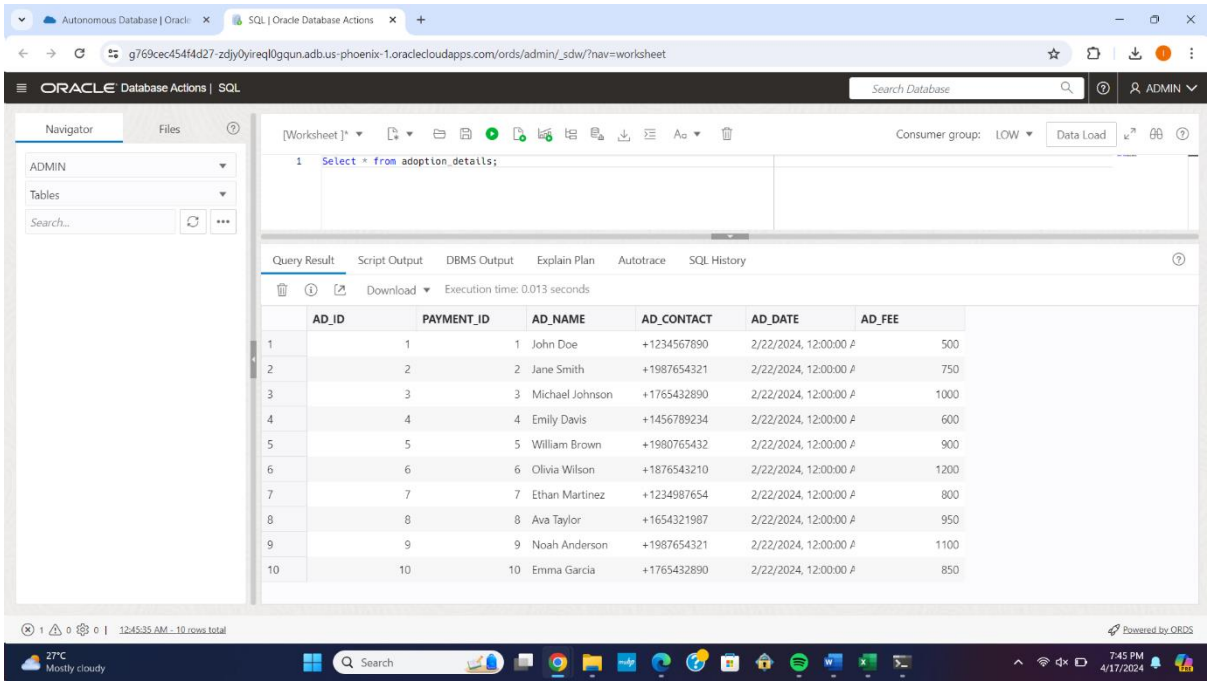


Creating a table named adoption_details

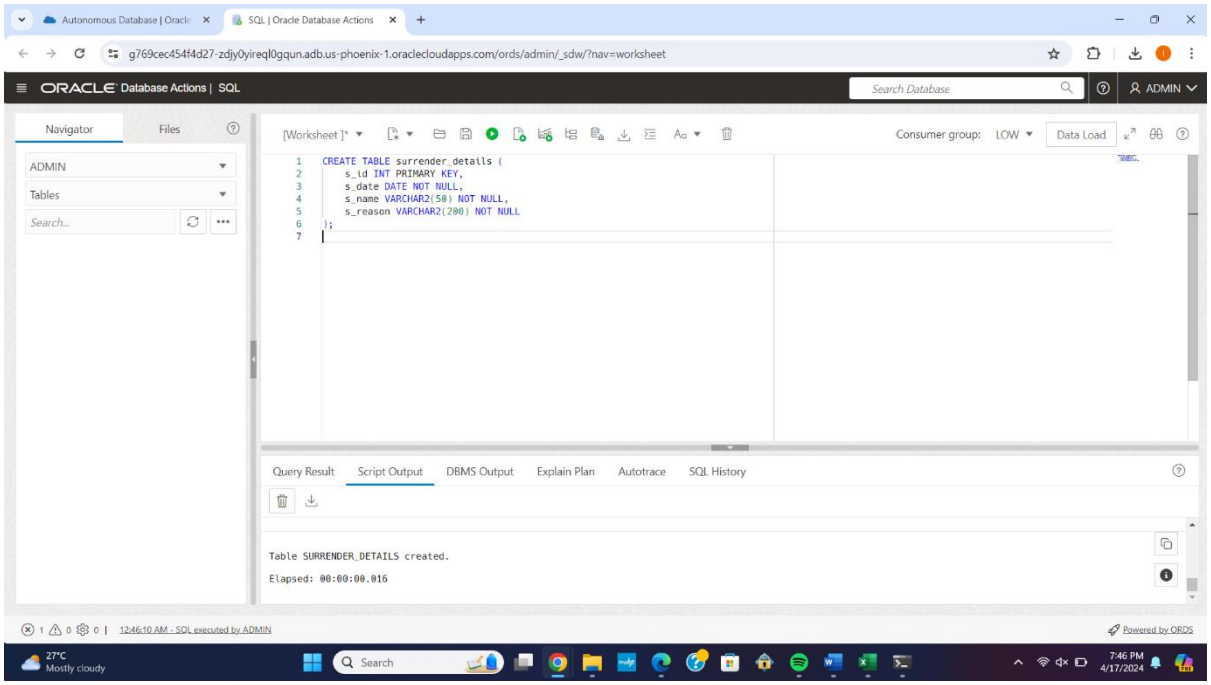


Inserting values to adoption_details

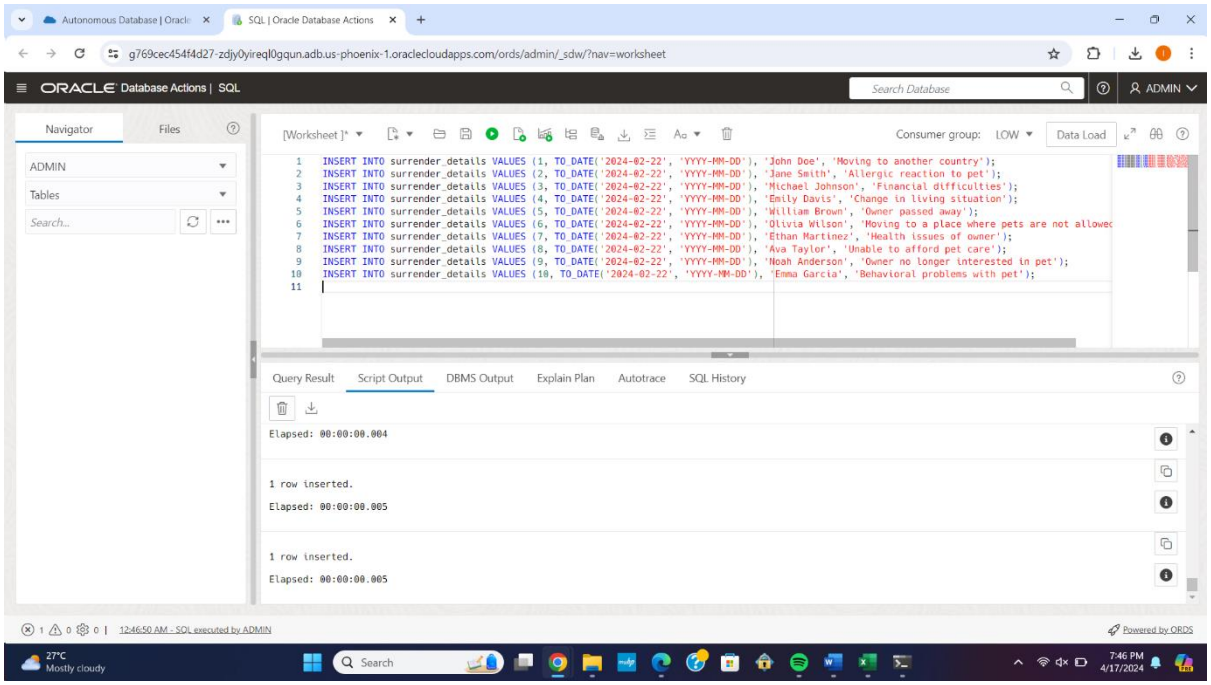


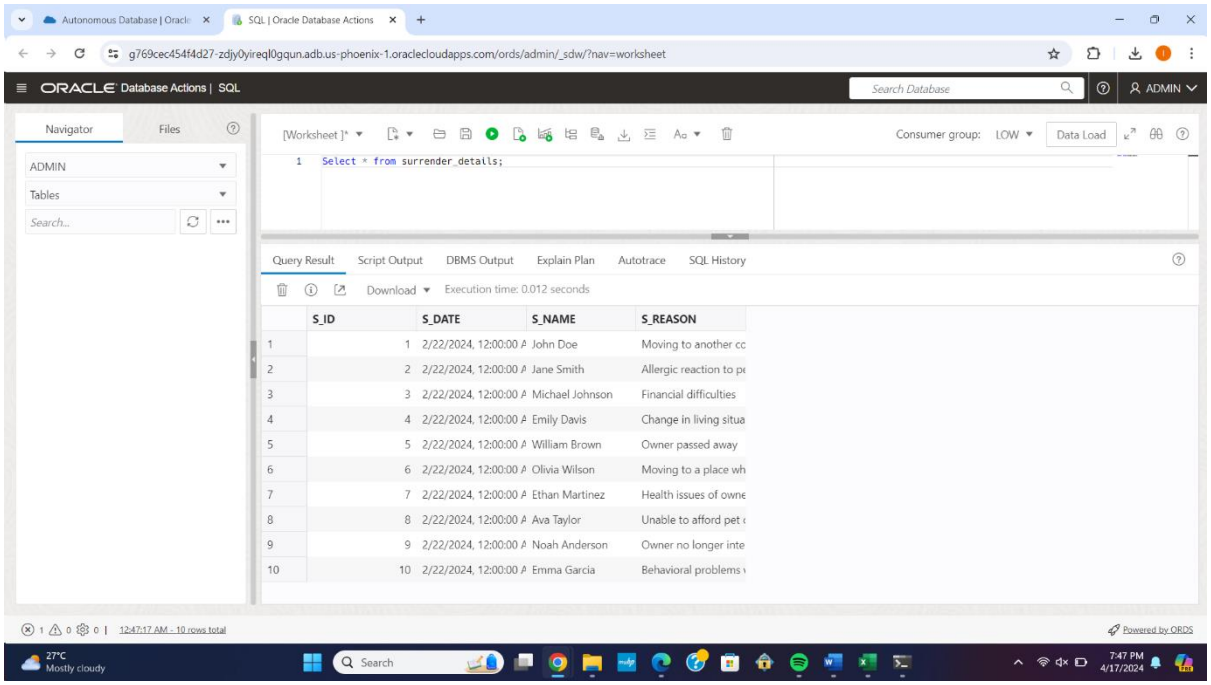


Creating a table named surrender_details

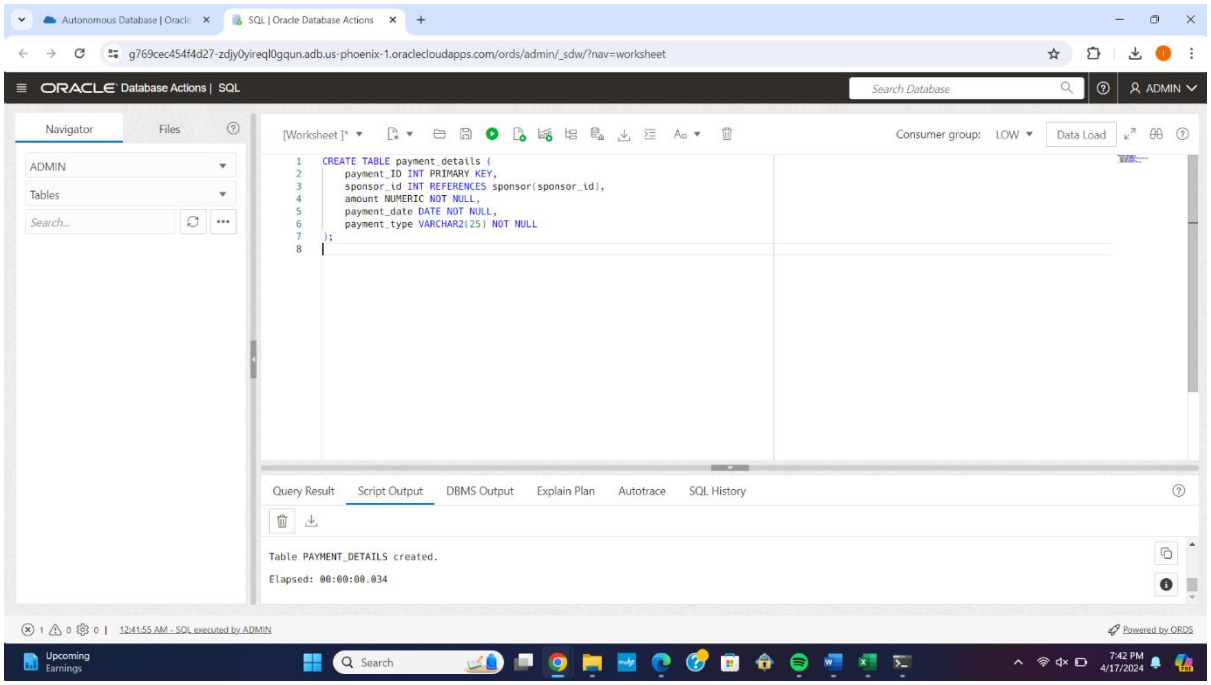


Inserting values to surrender_details

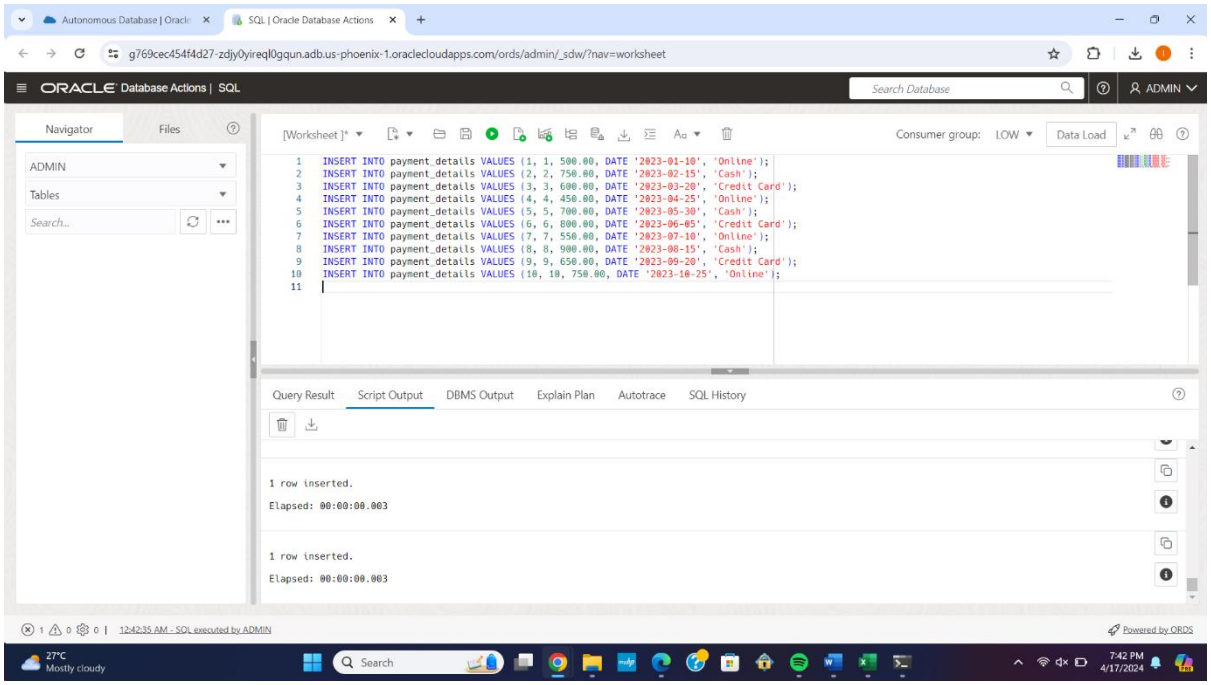


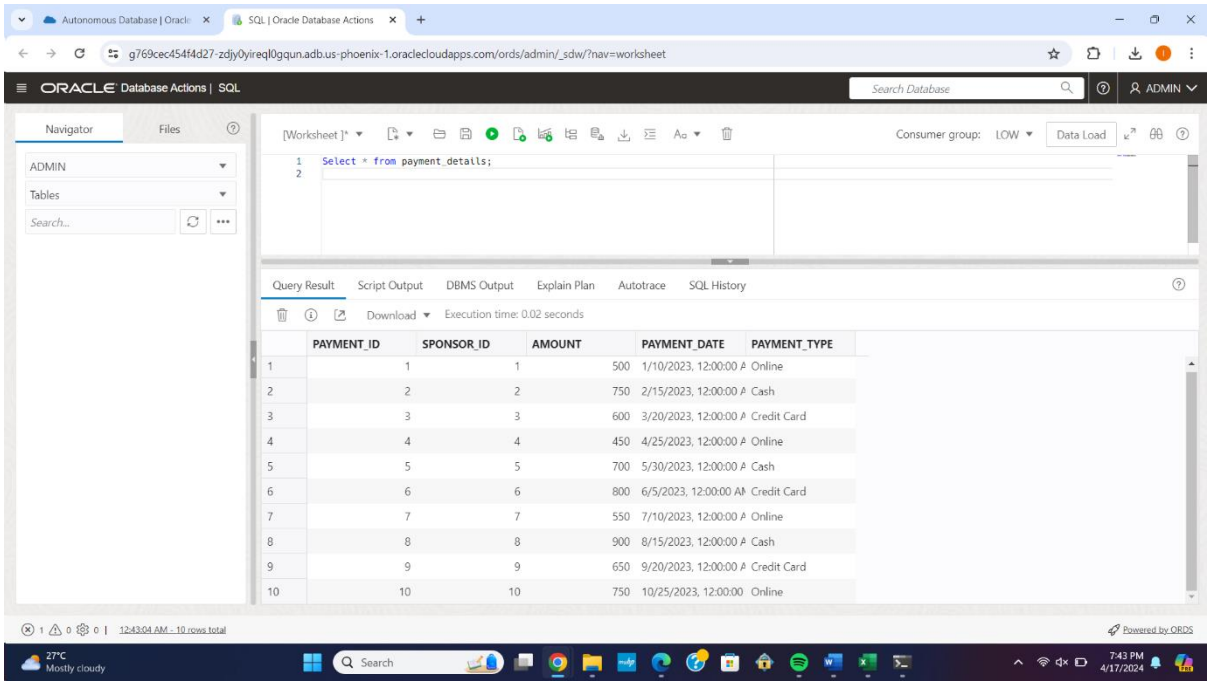


Creating a table named payment_details

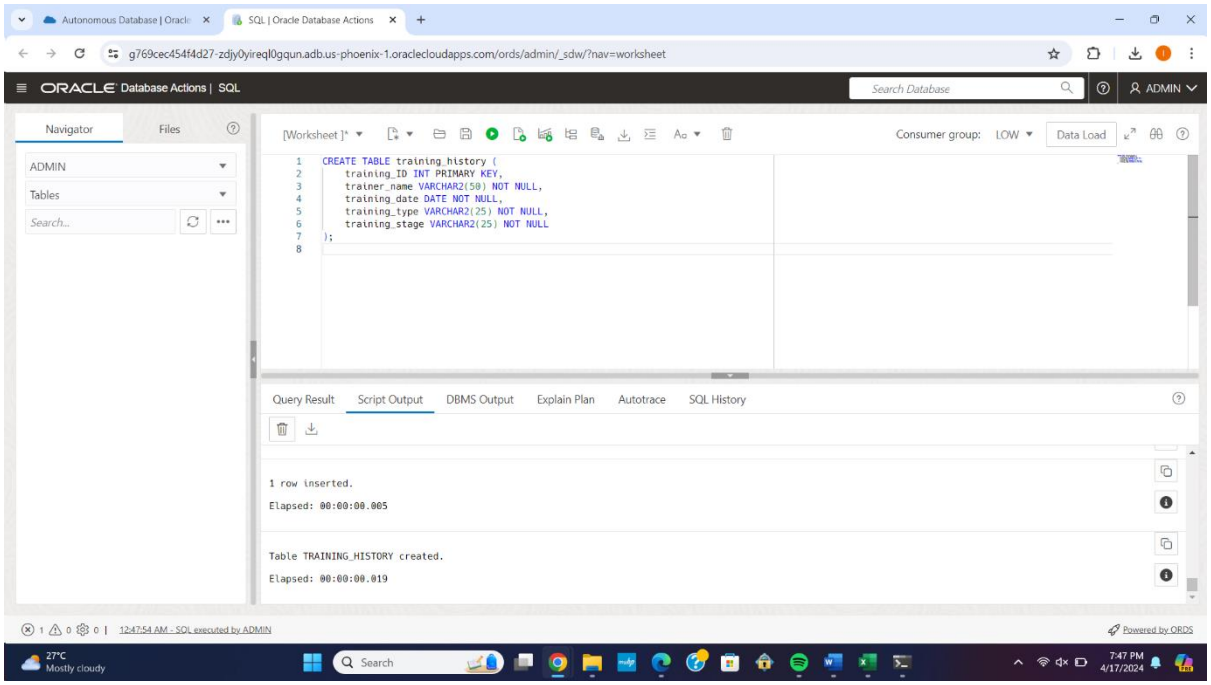


Inserting values to payment_details

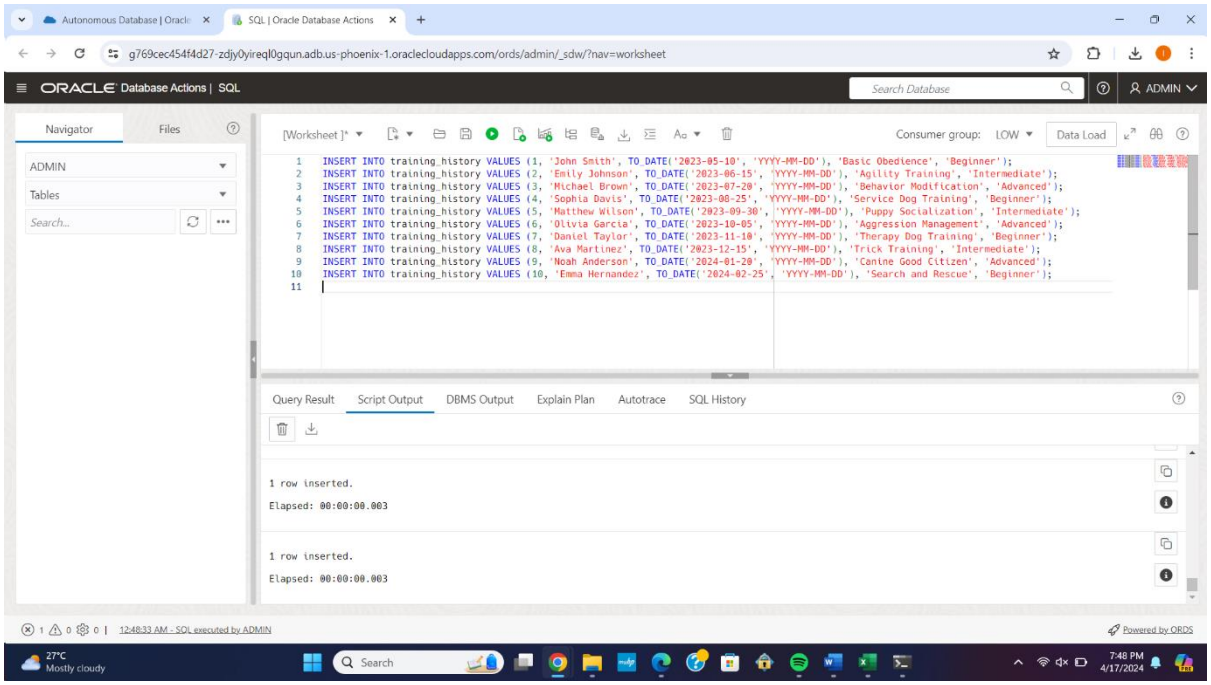


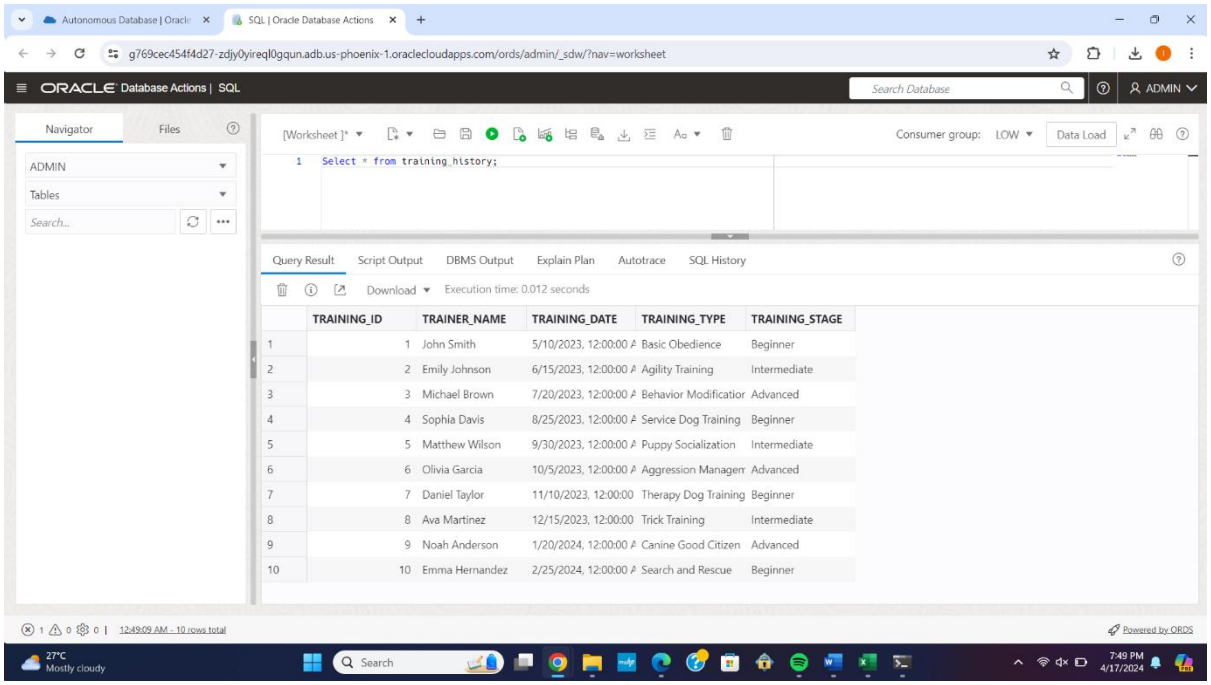


Creating a table named training_history

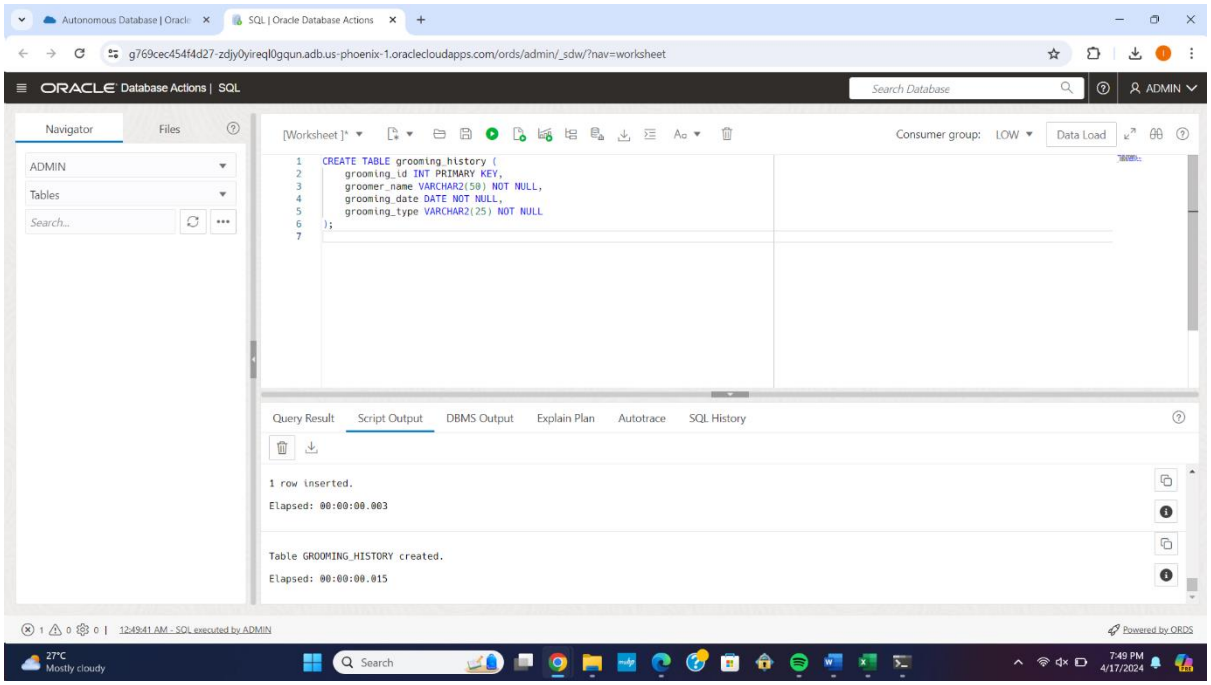


Inserting values to training_history

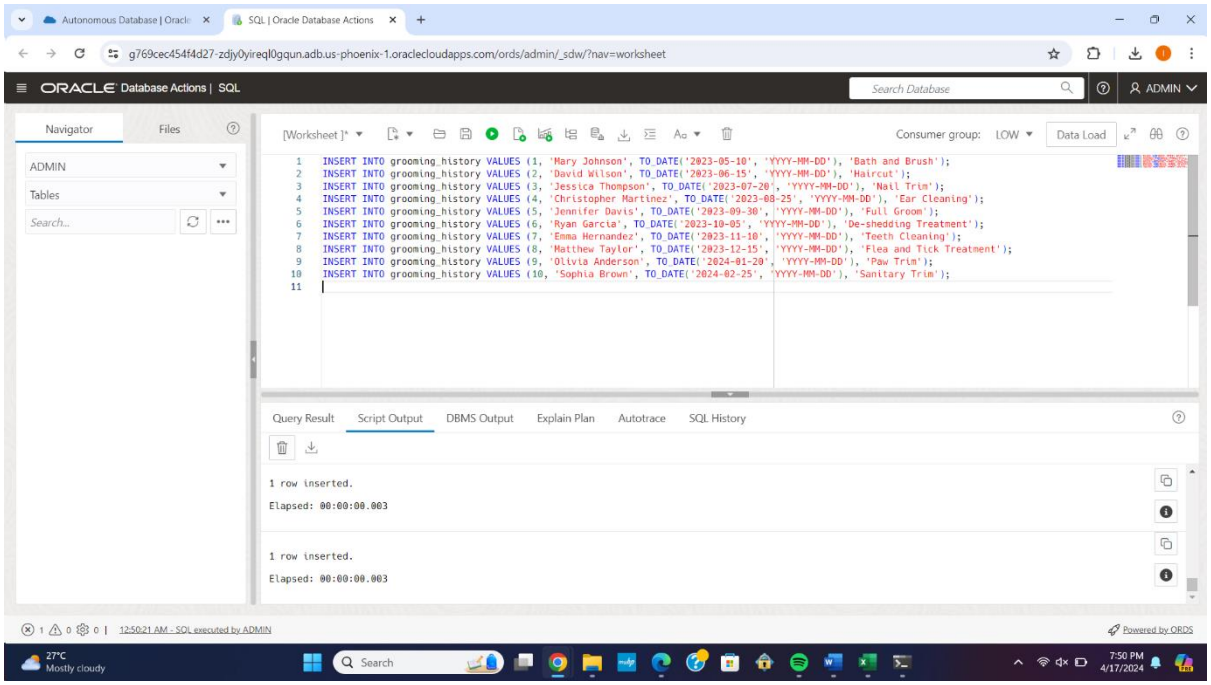


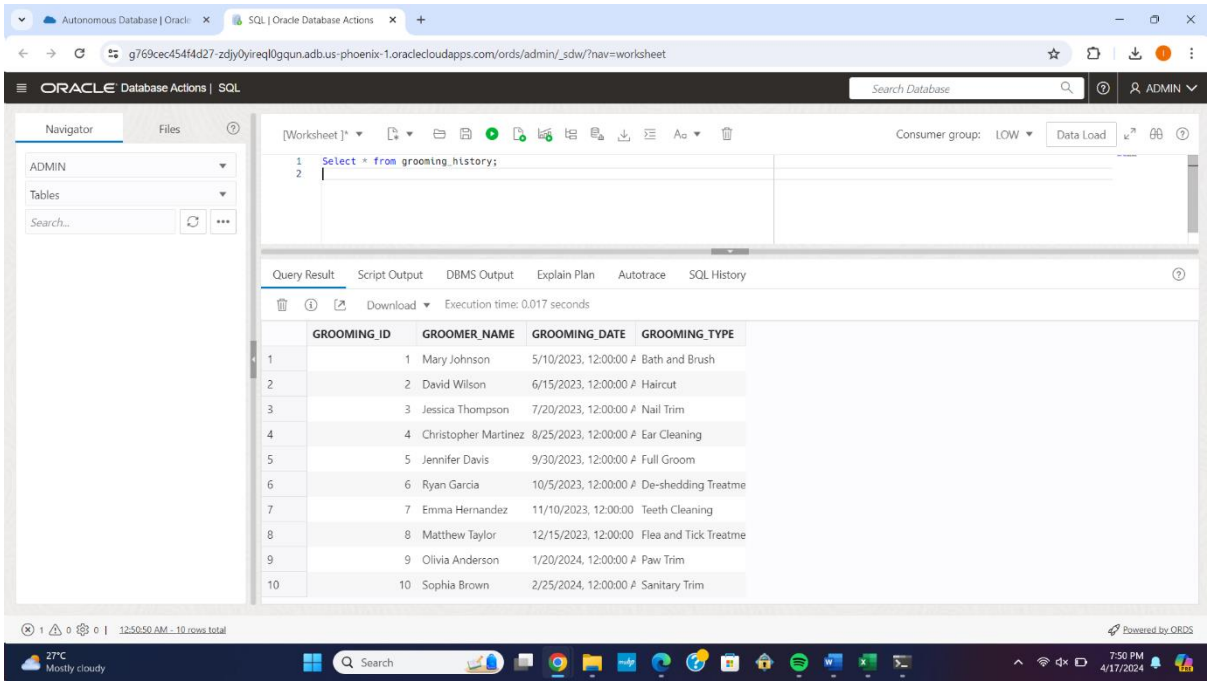


Creating a table named grooming_history

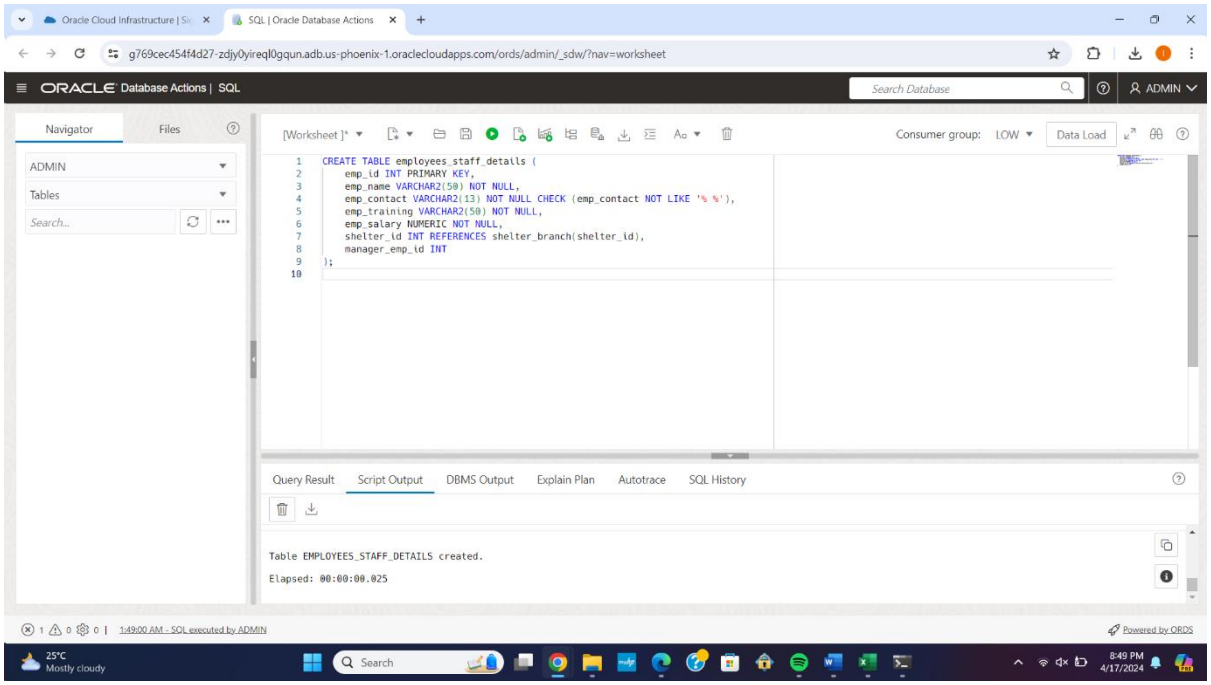


Inserting values to grooming_history

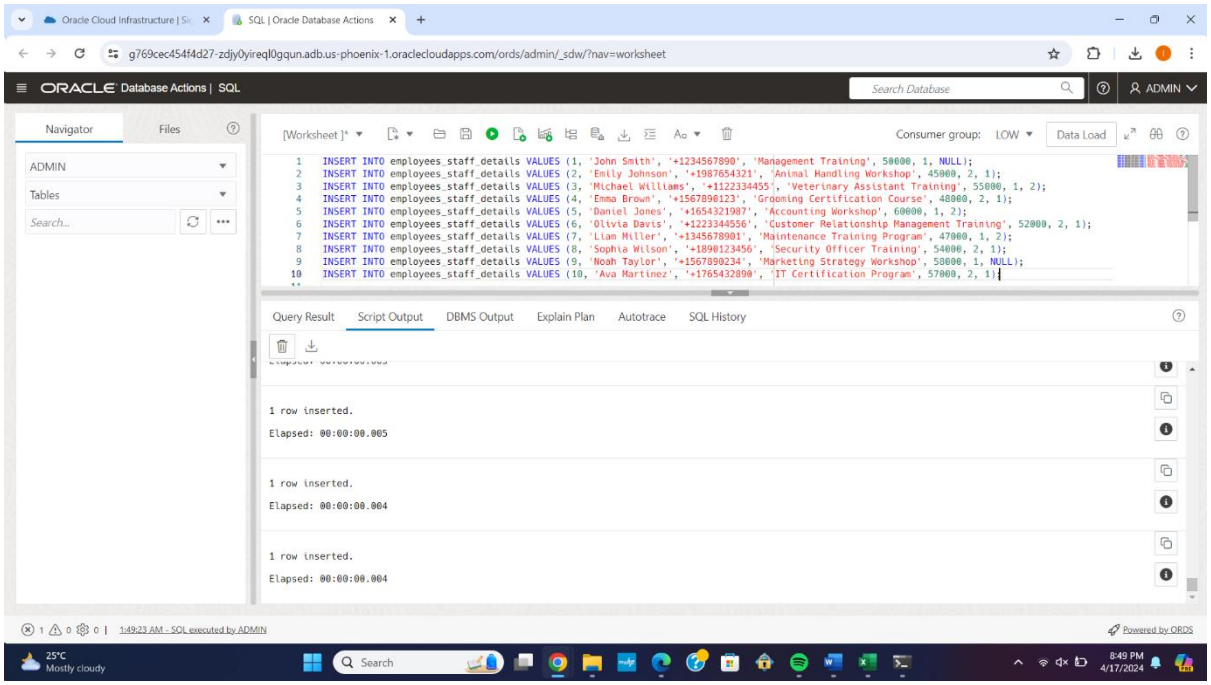


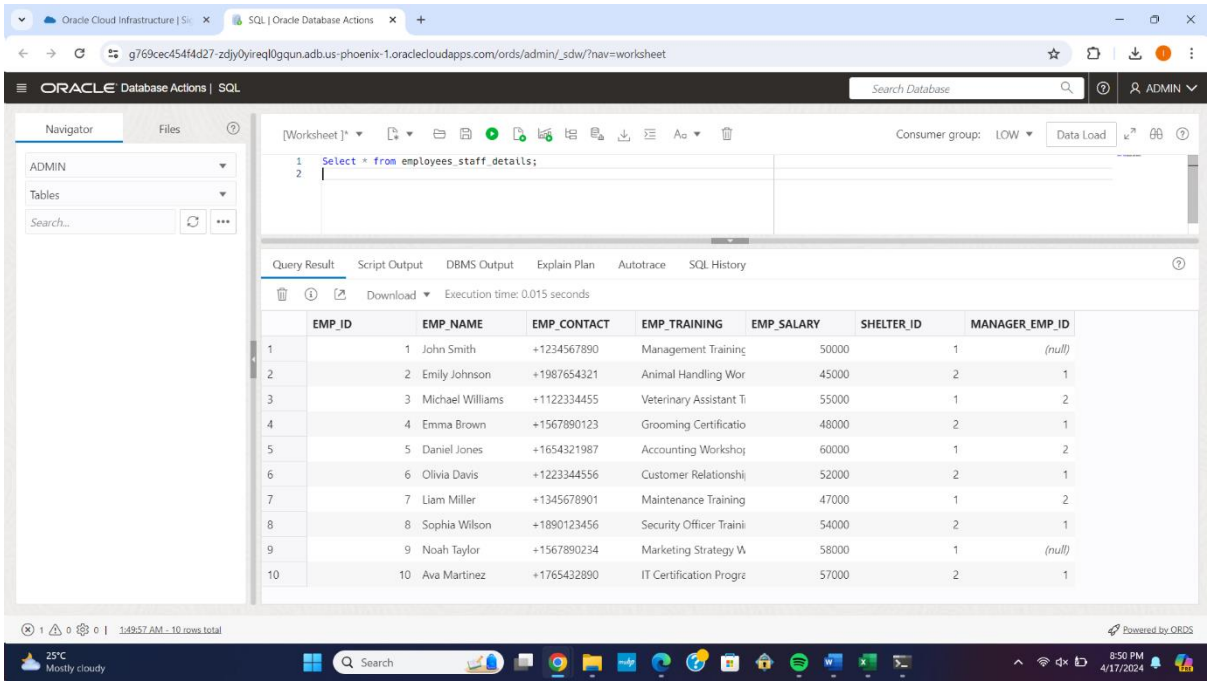


Creating a table named employees_staff_details

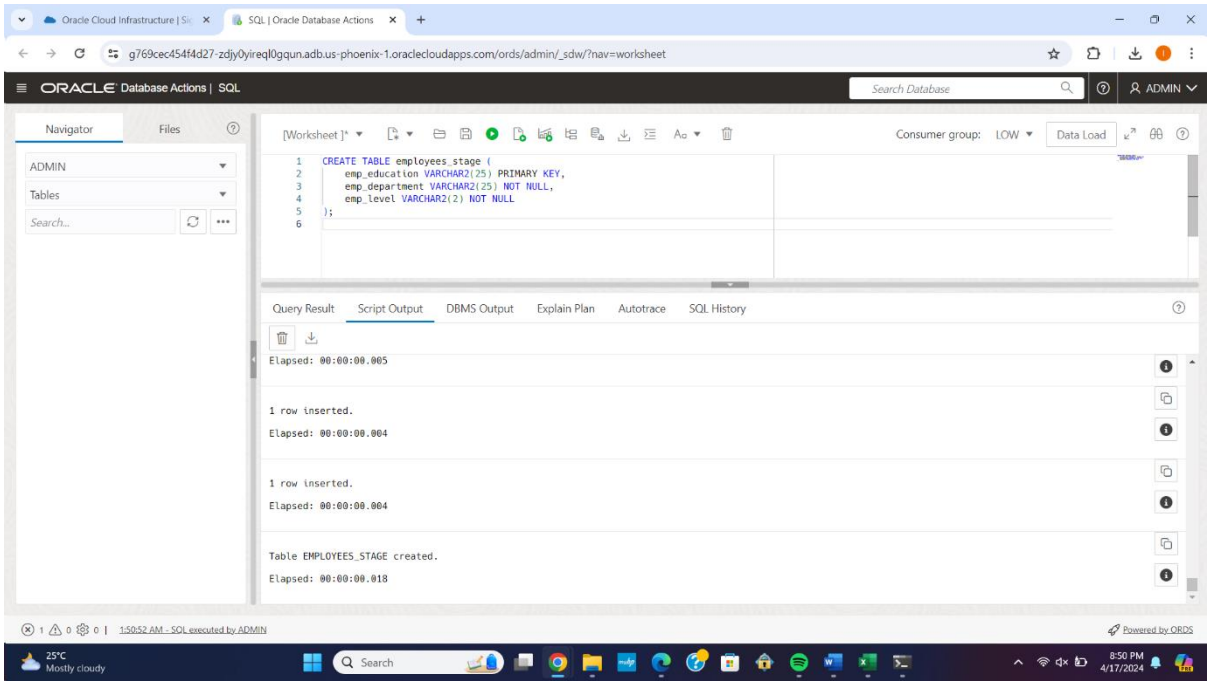


Inserting values to employees_staff_details

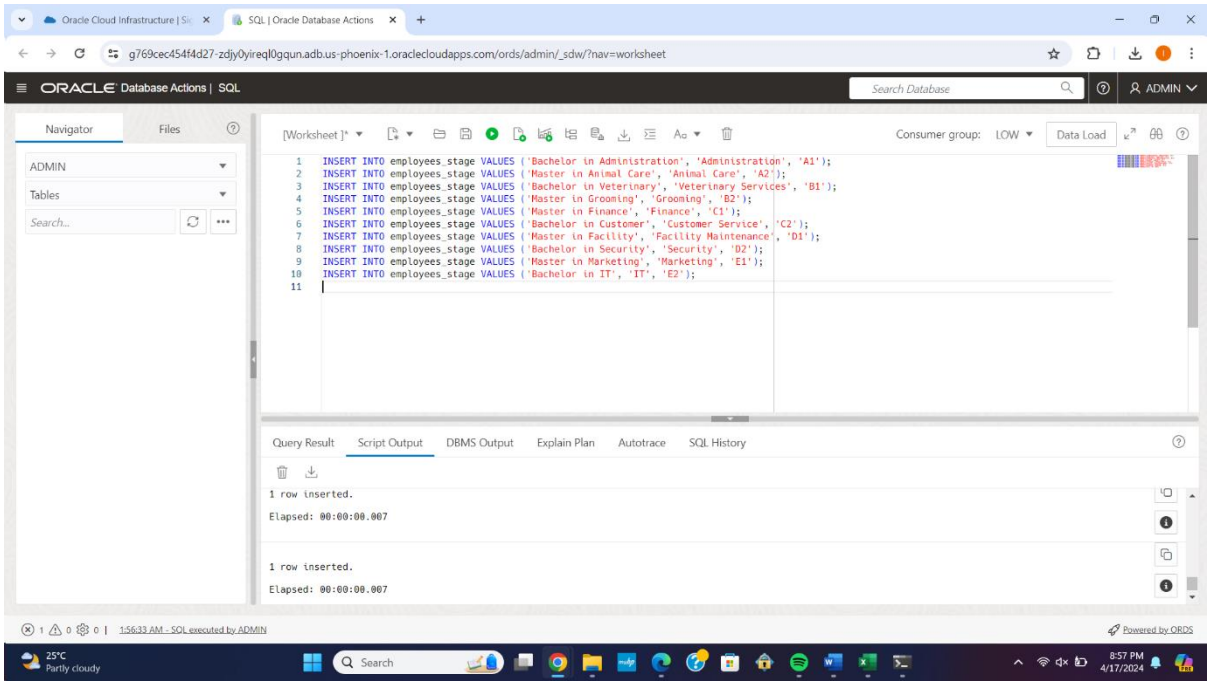


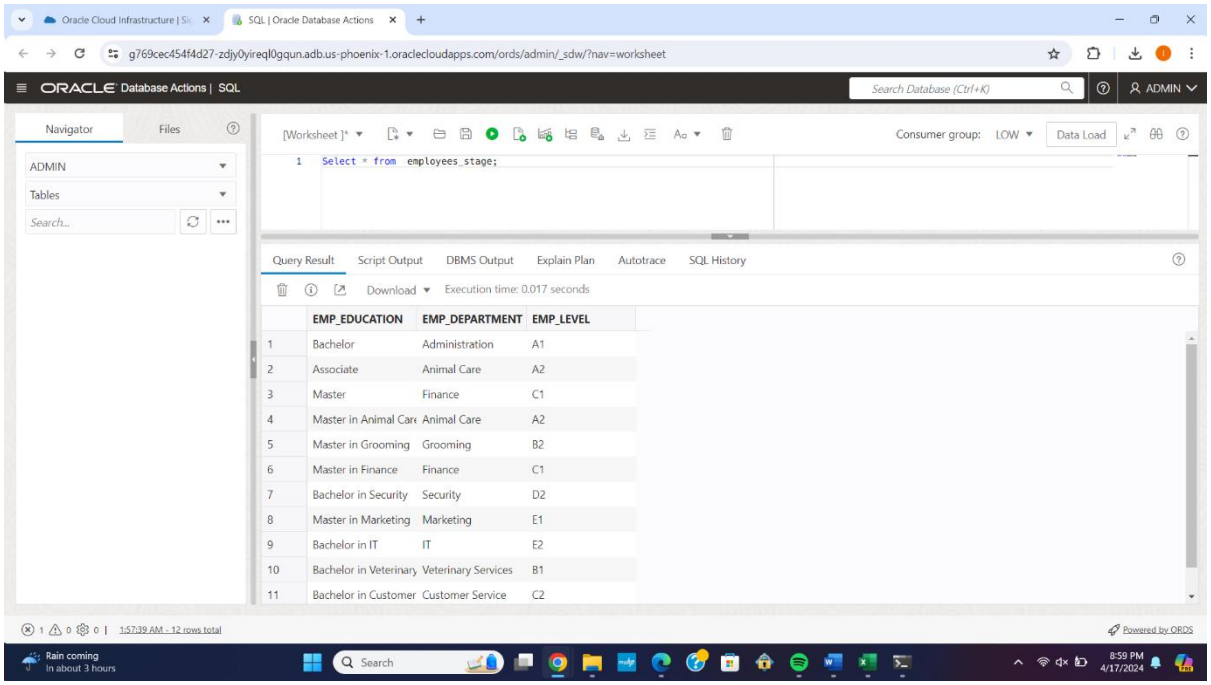


Creating a table named employees_stage(New)

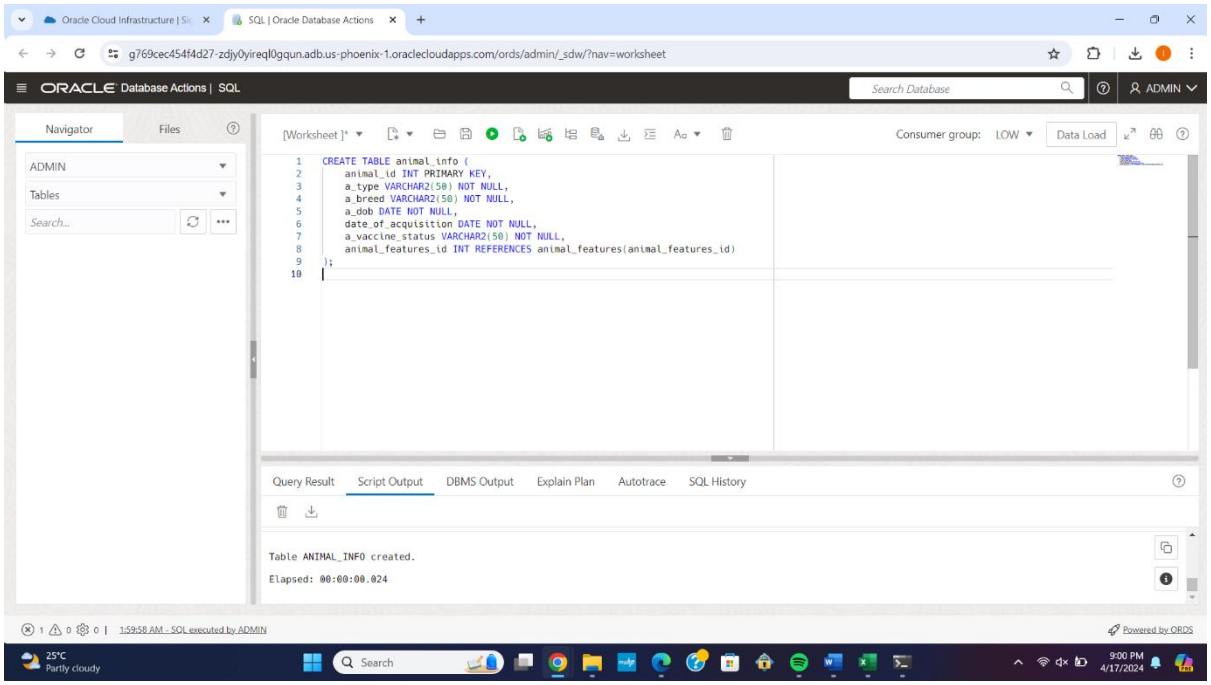


Inserting values to employees_stage(New)

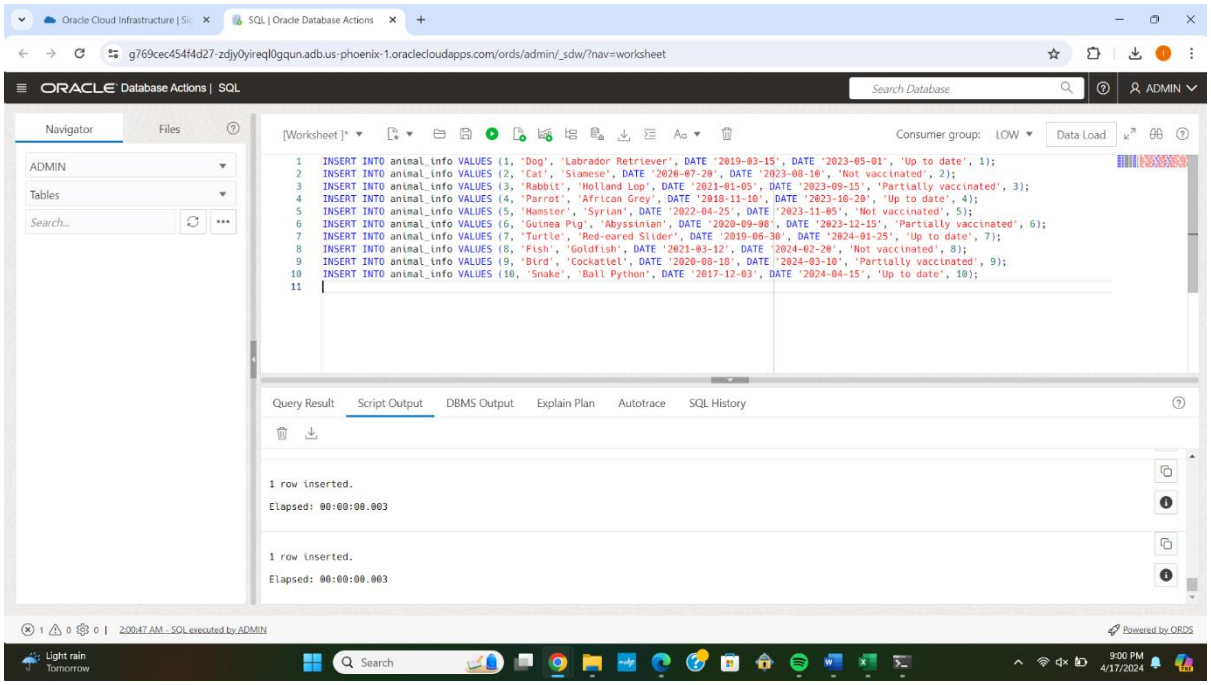


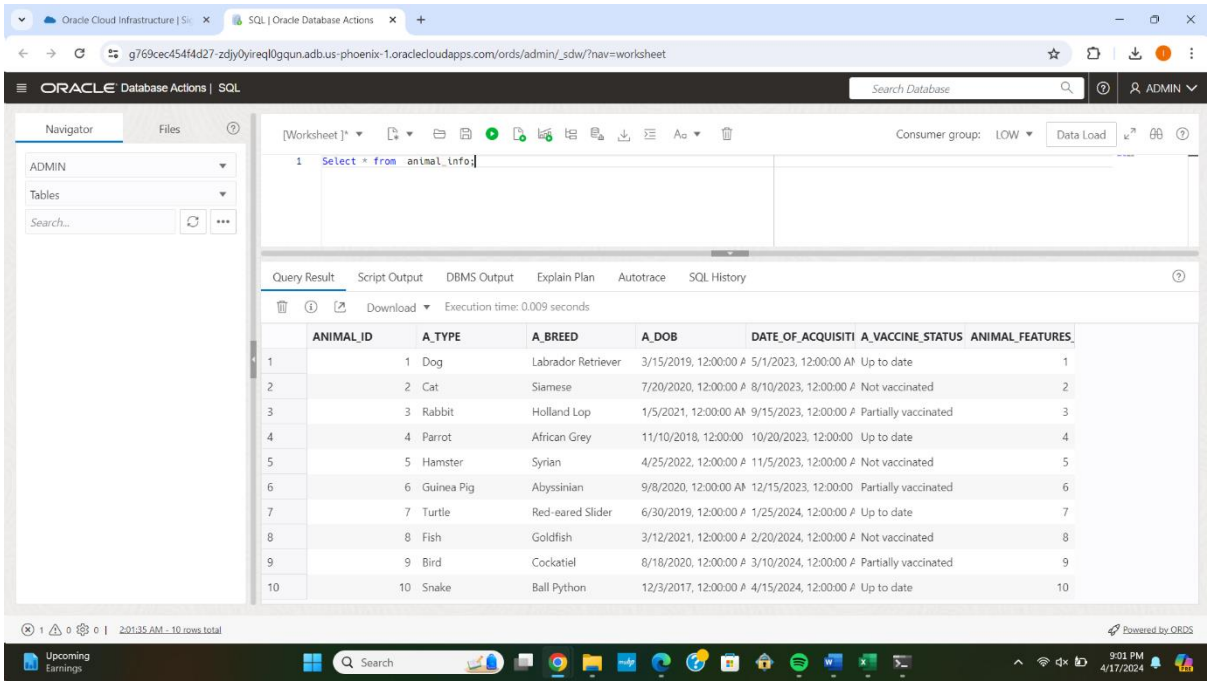


Creating a table named animal_info

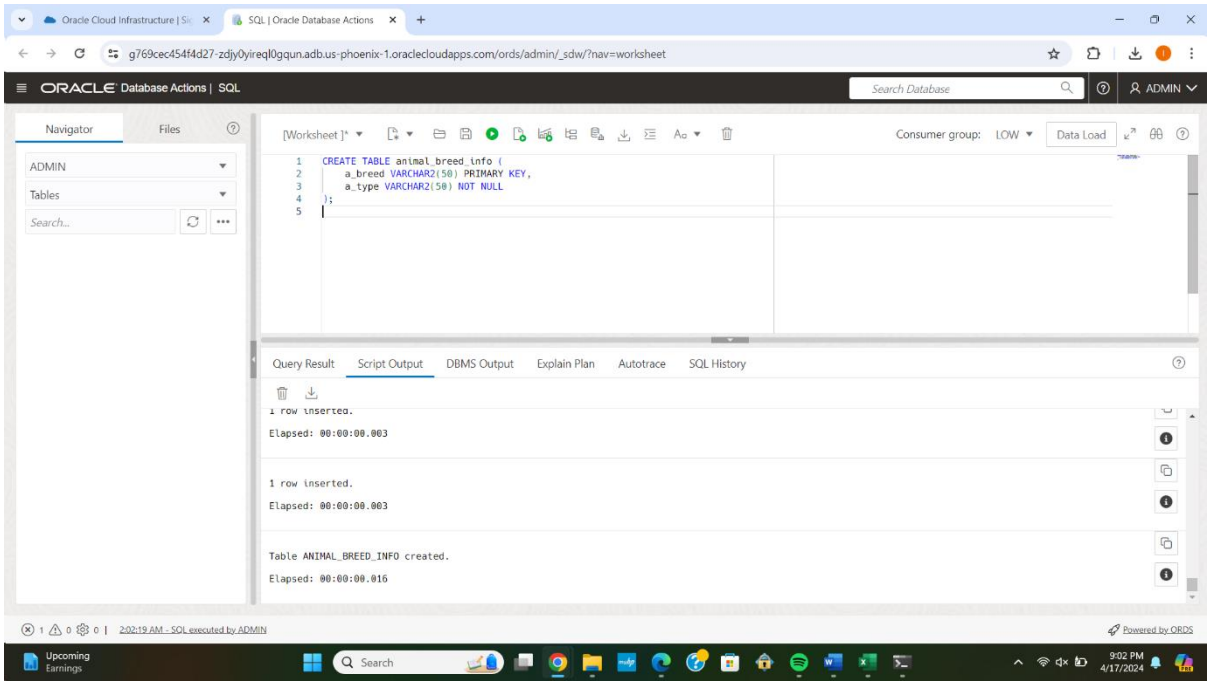


Inserting values to animal_info

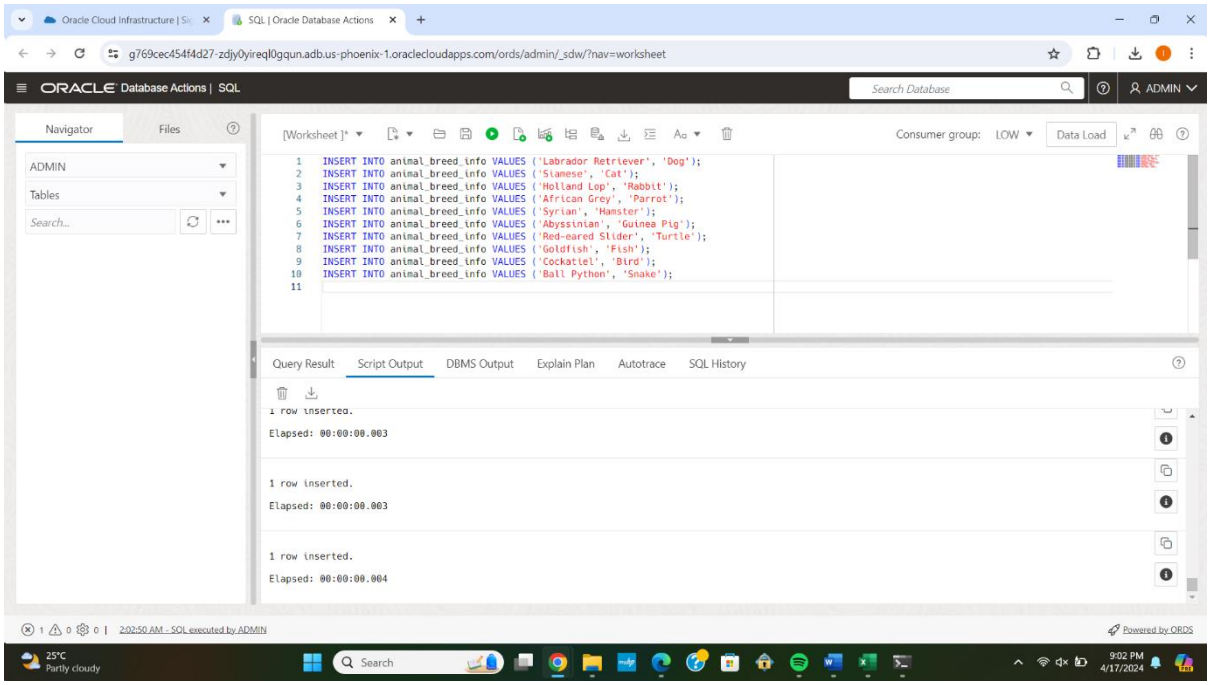


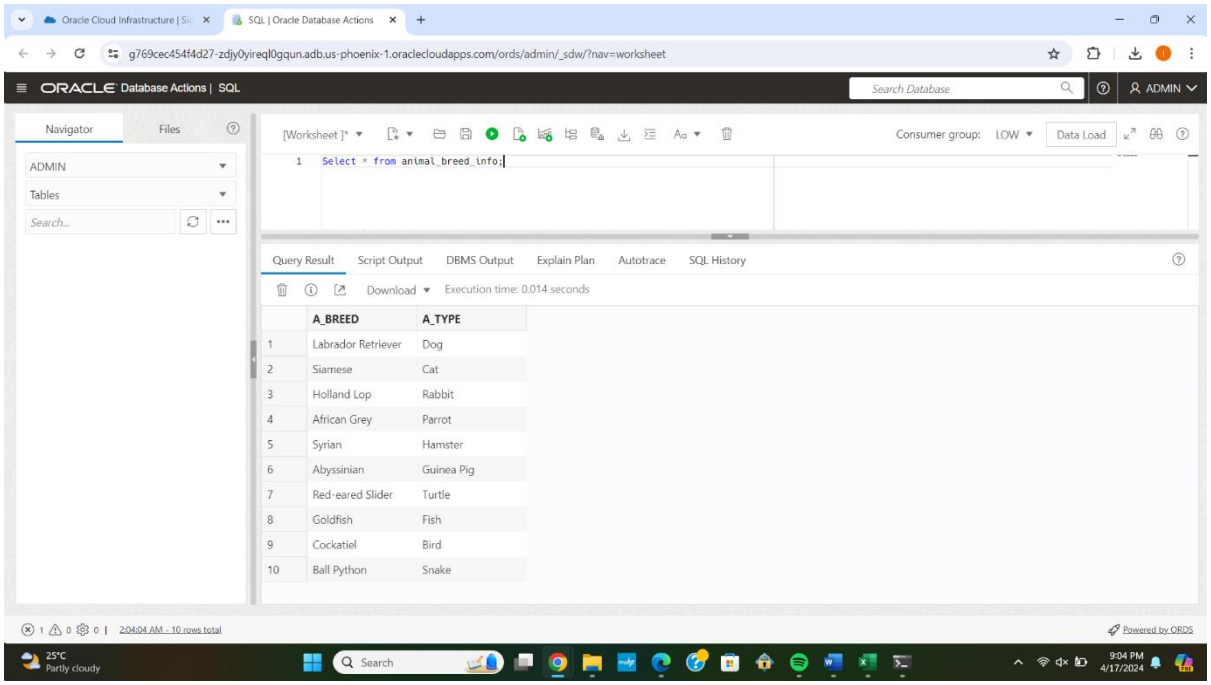


Creating a table named animal_breed_info(New)

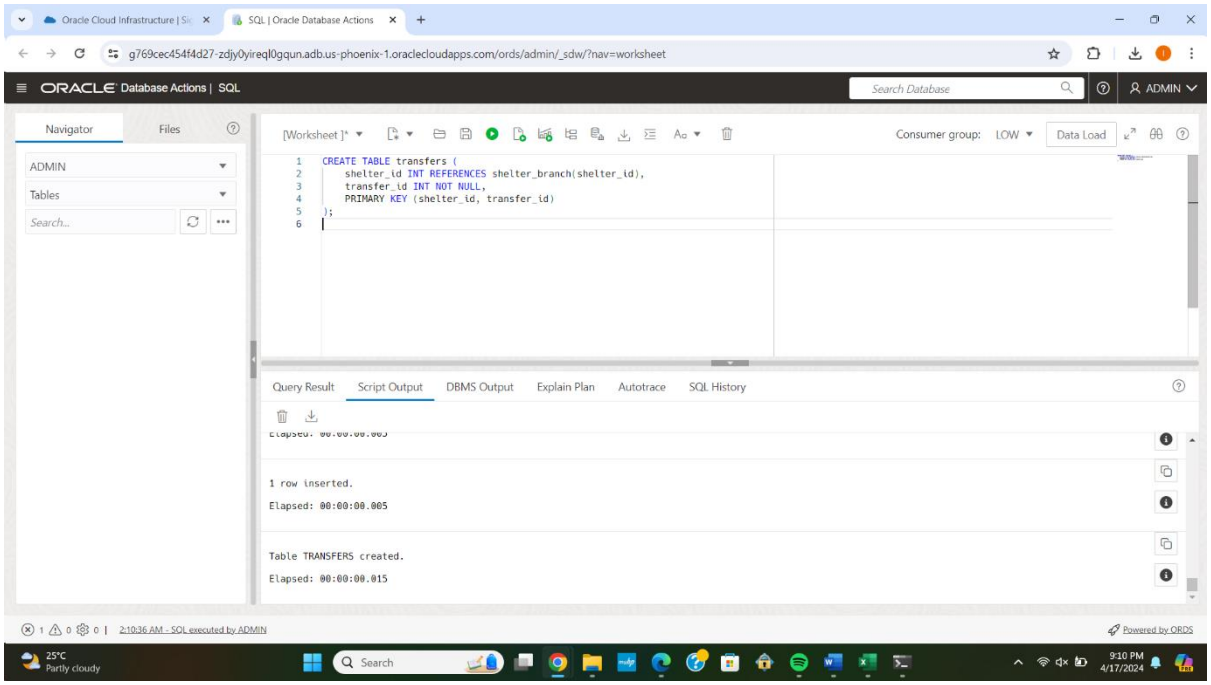


Inserting values to animal_breed_info(New)

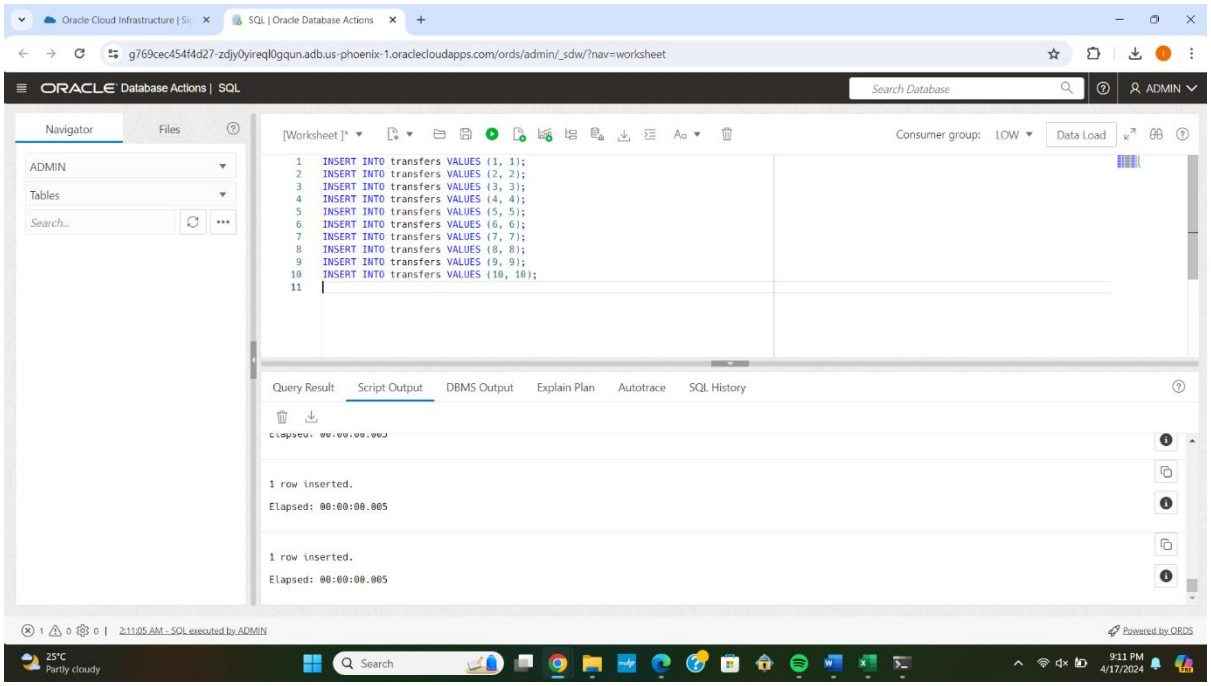


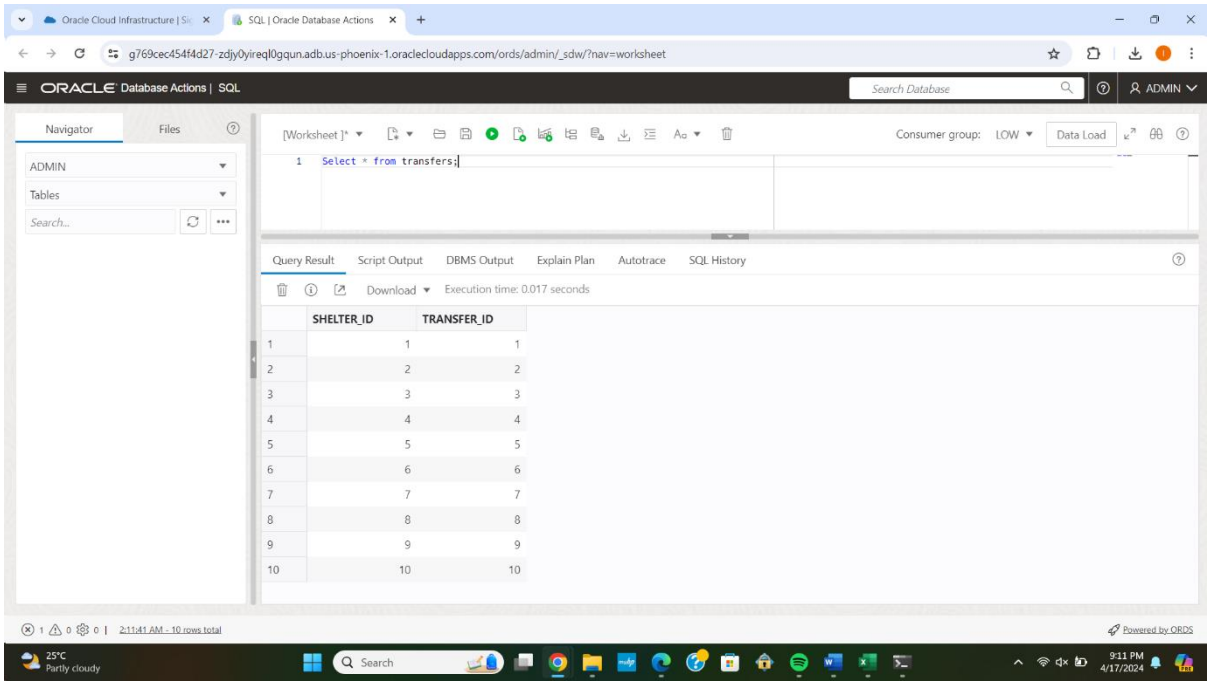


Creating a table named transfers

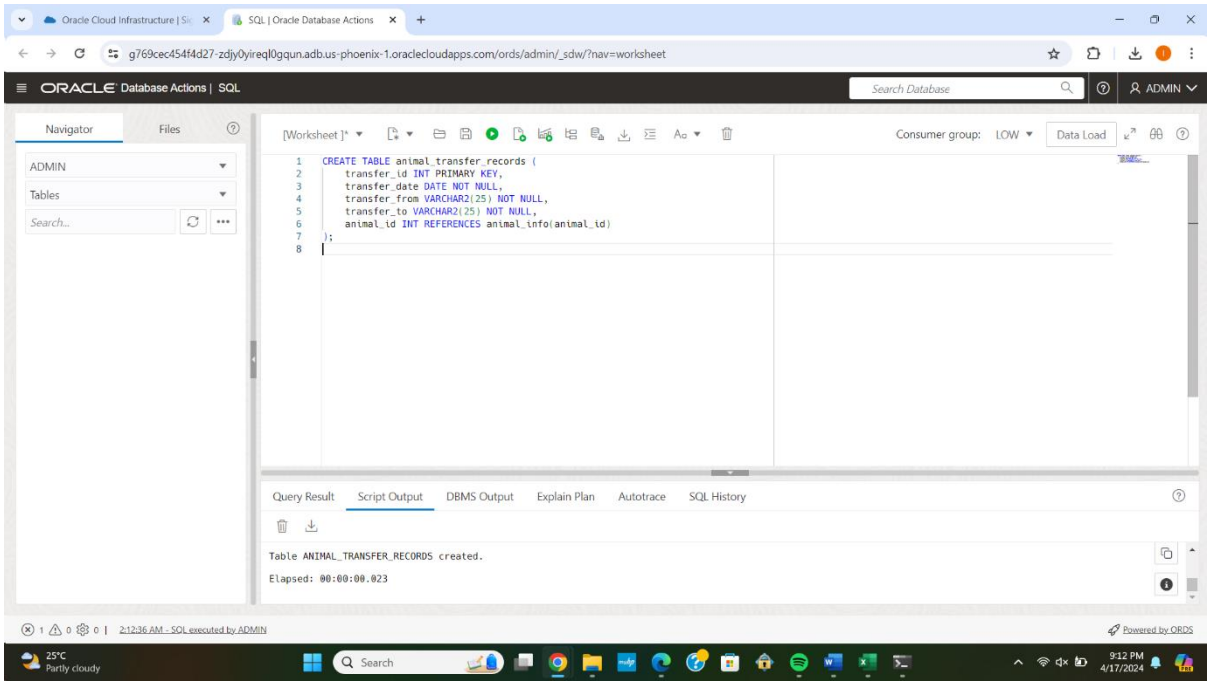


Inserting values to transfers

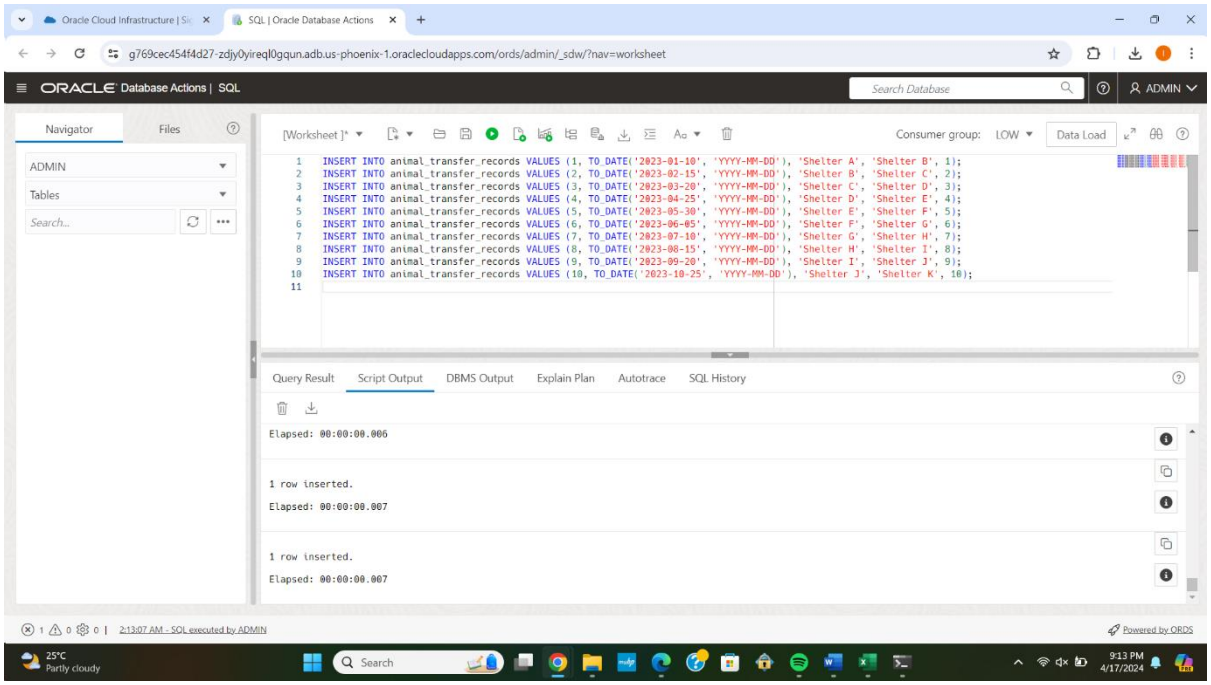


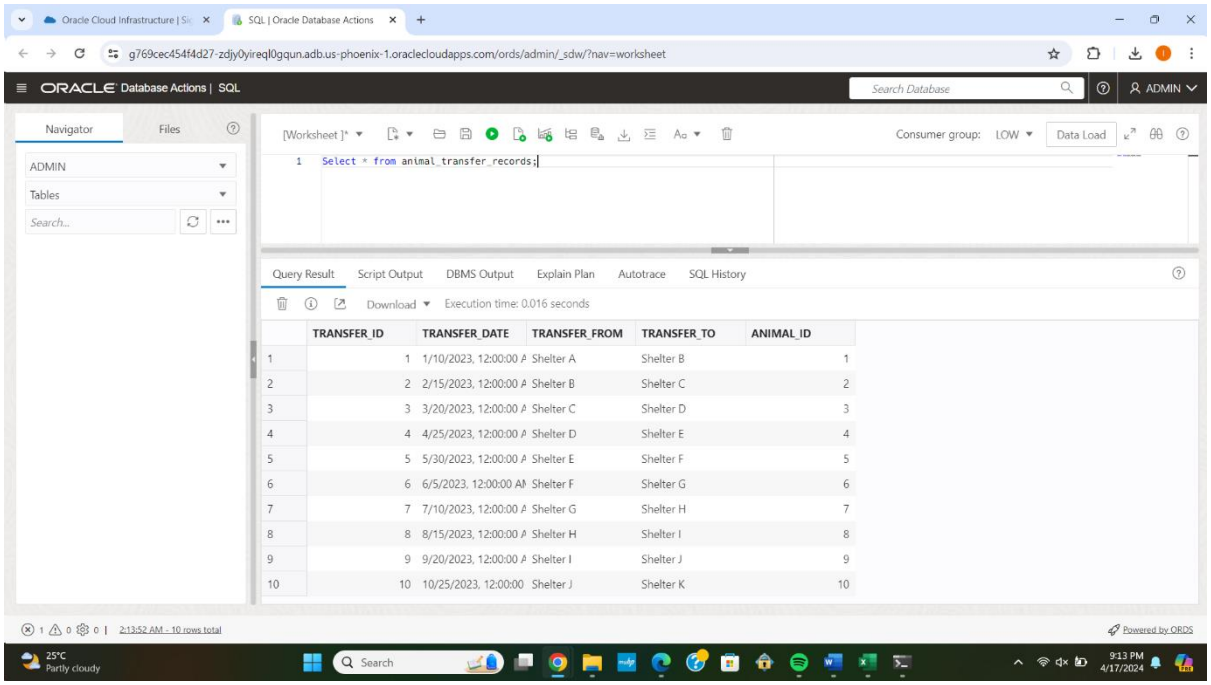


Creating a table named animal_transfer_records

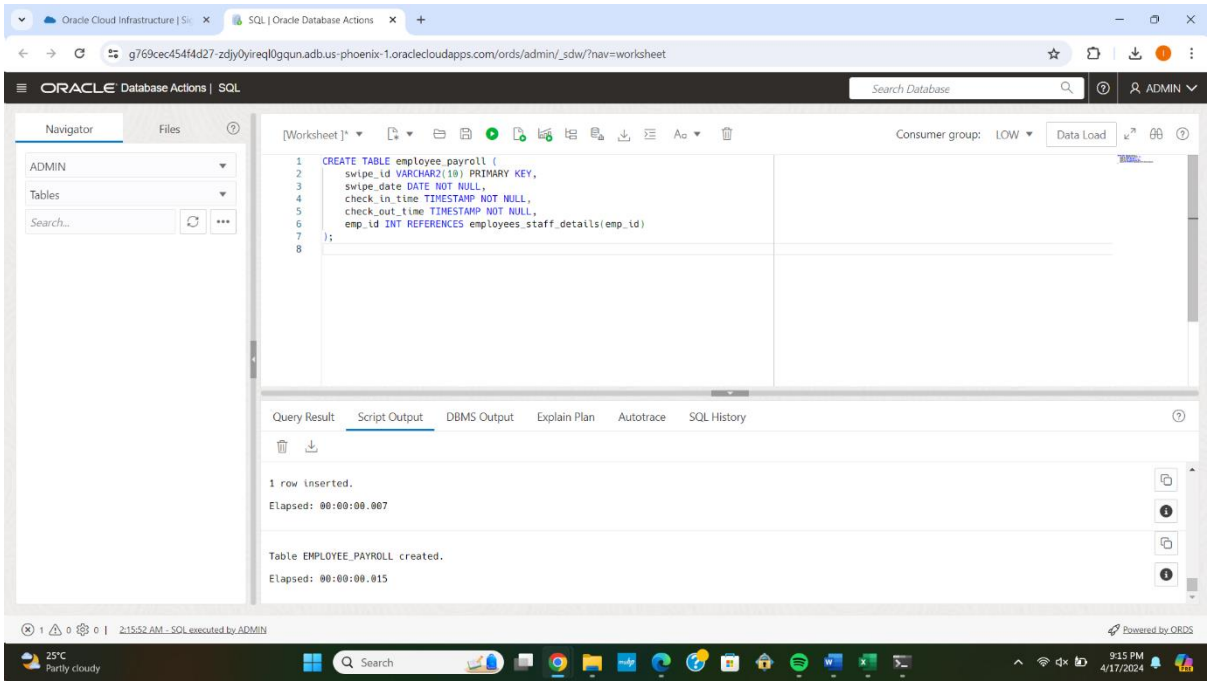


Inserting values to animal_transfer_records

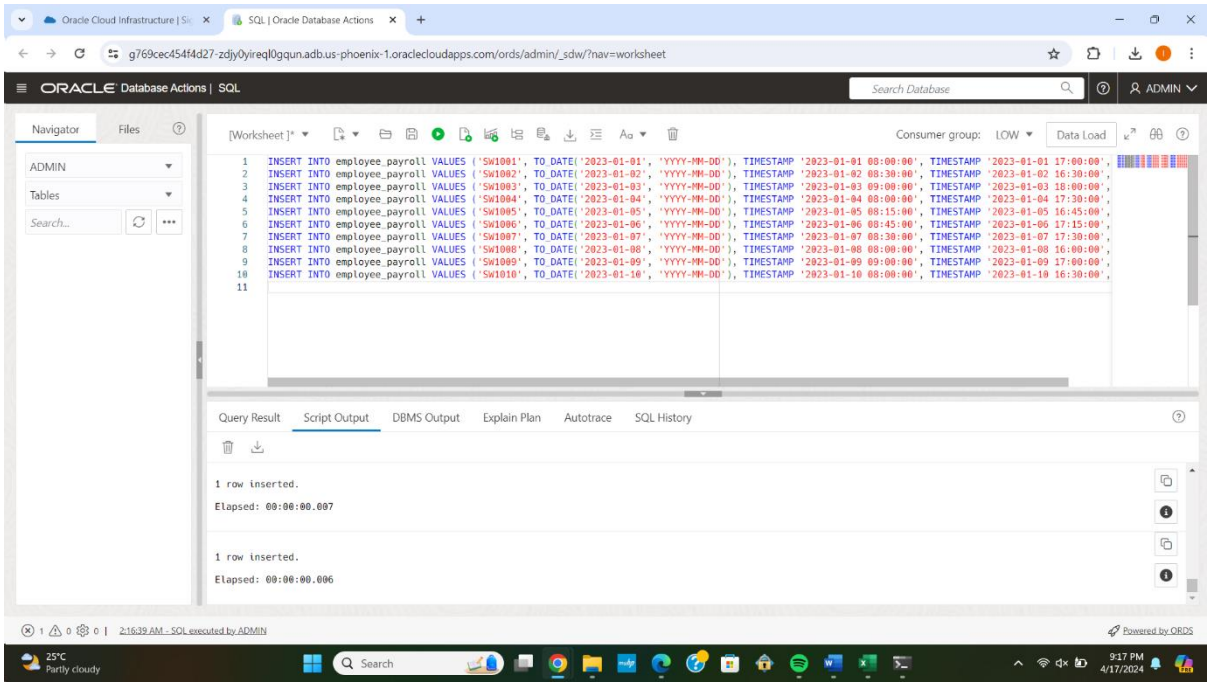


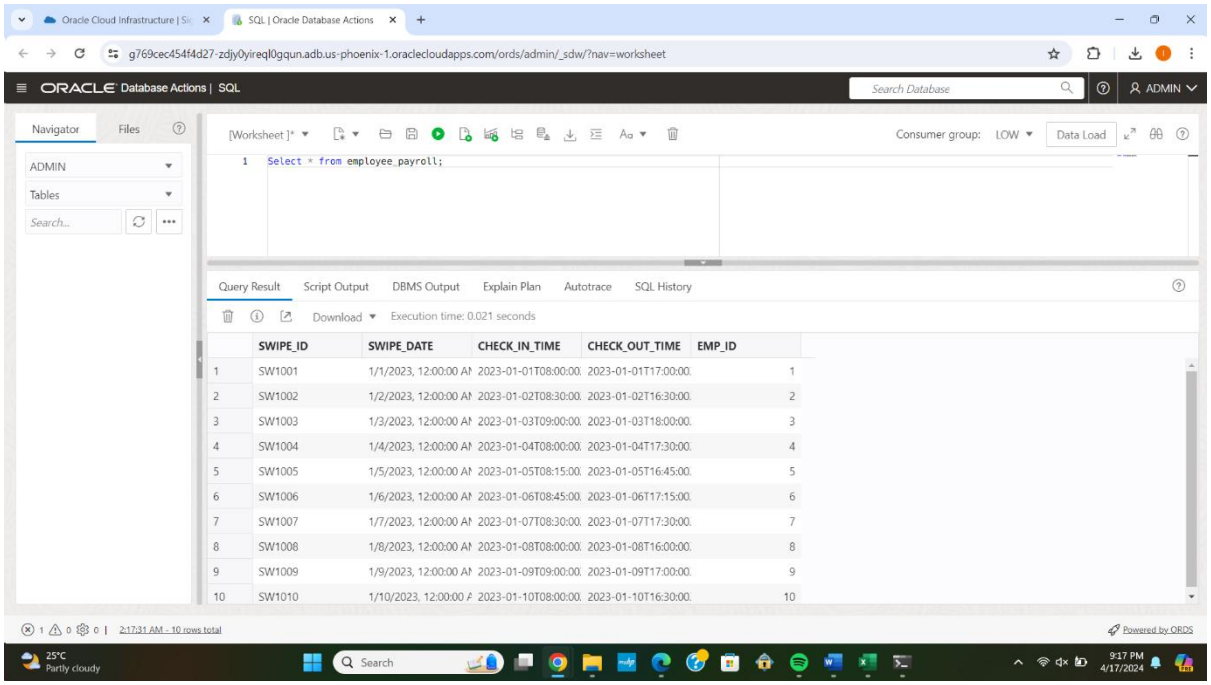


Creating a table named employee_payroll

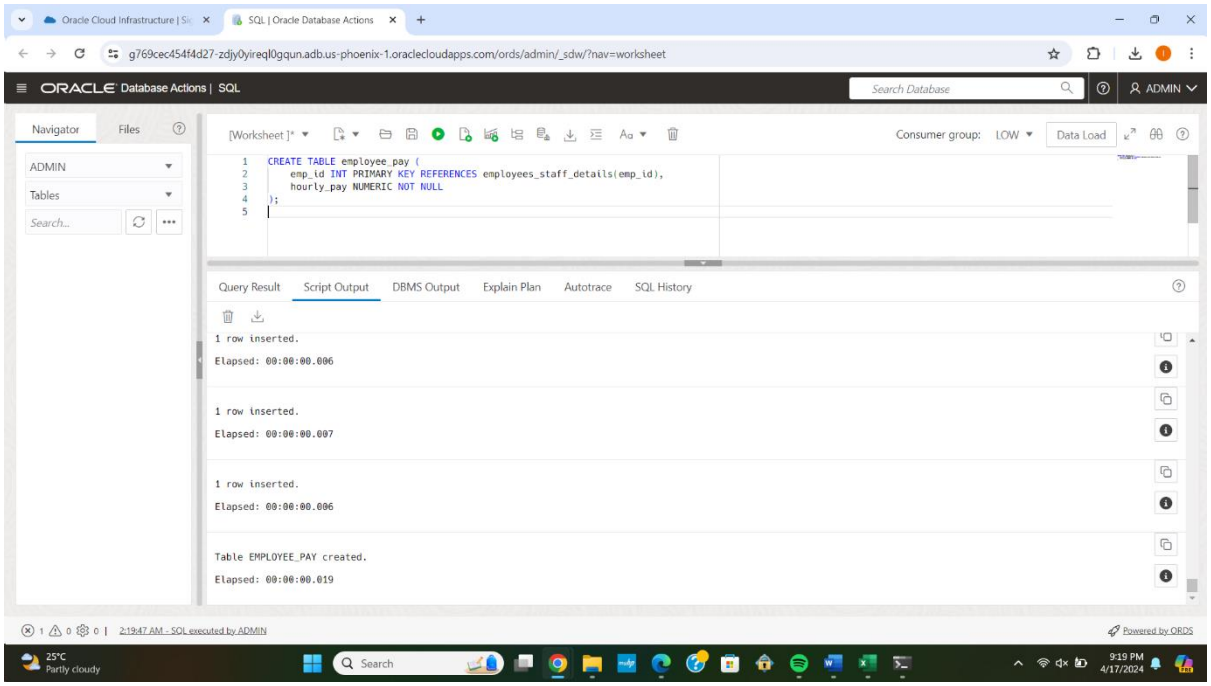


Inserting values to employee_payroll

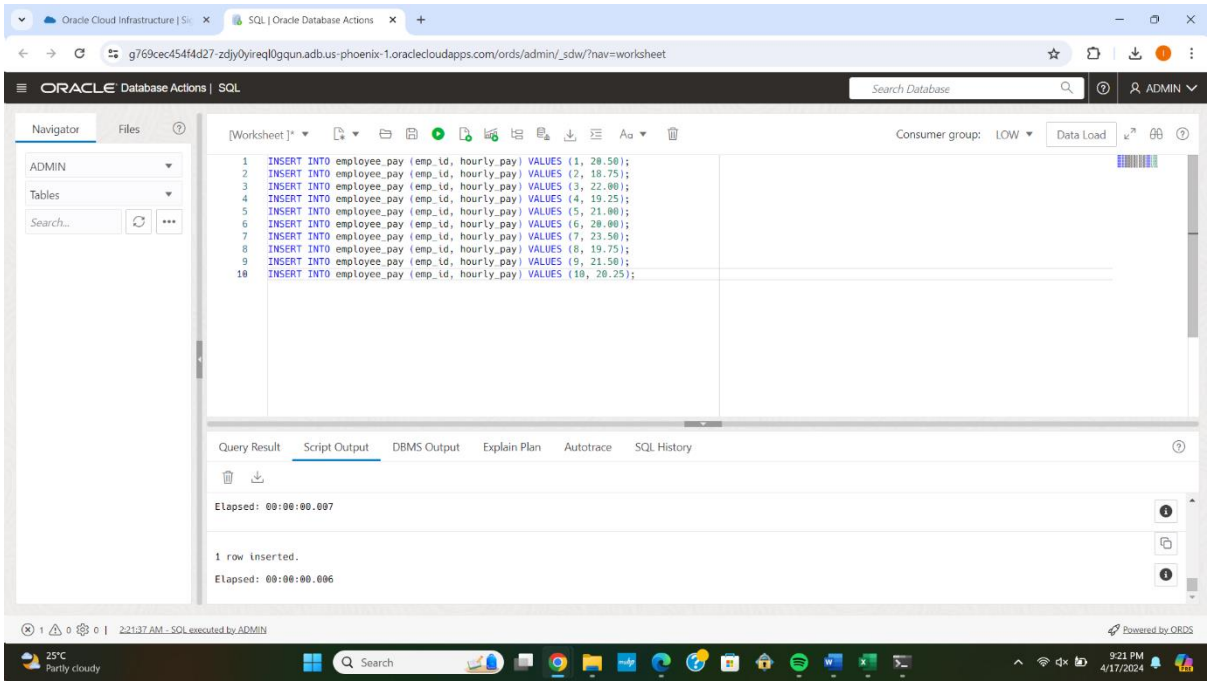


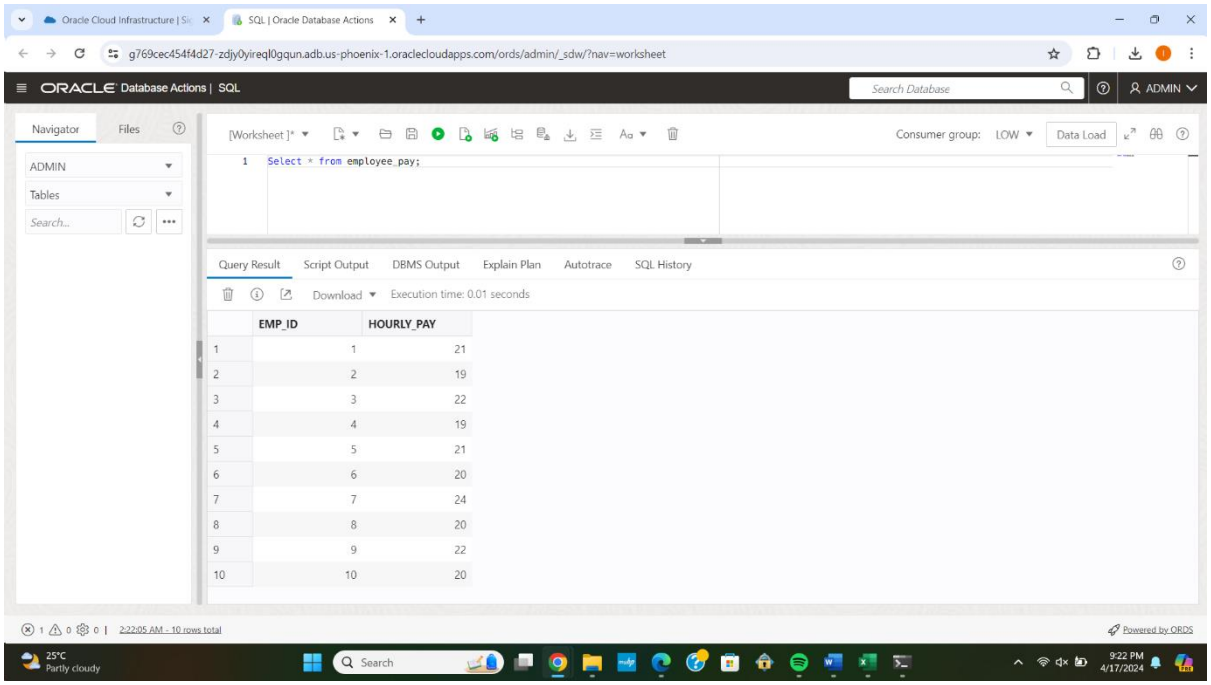


Creating a table named employee_pay(New)

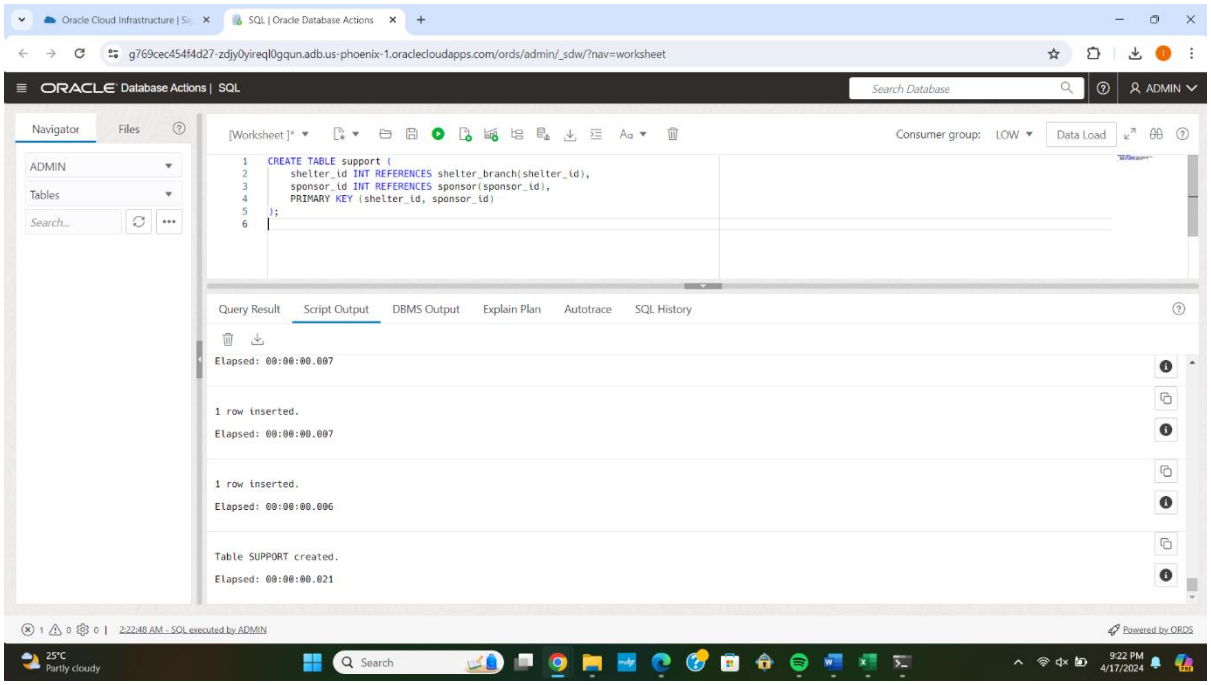


Inserting values to employee_pay(New)

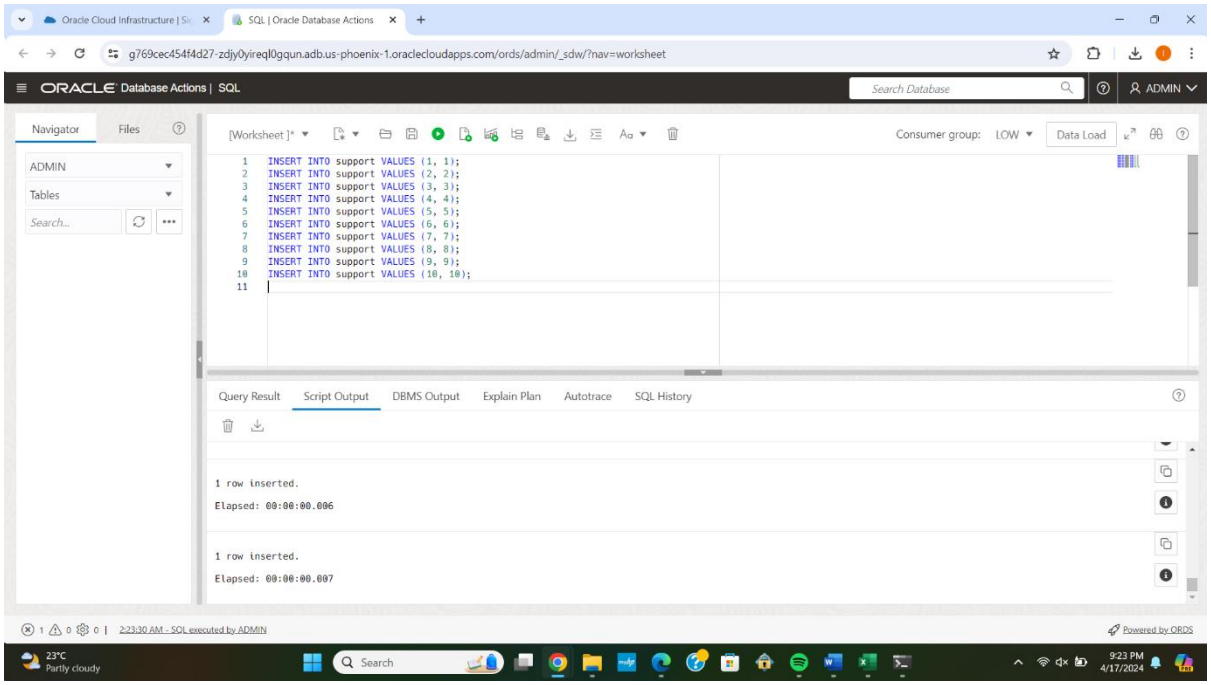


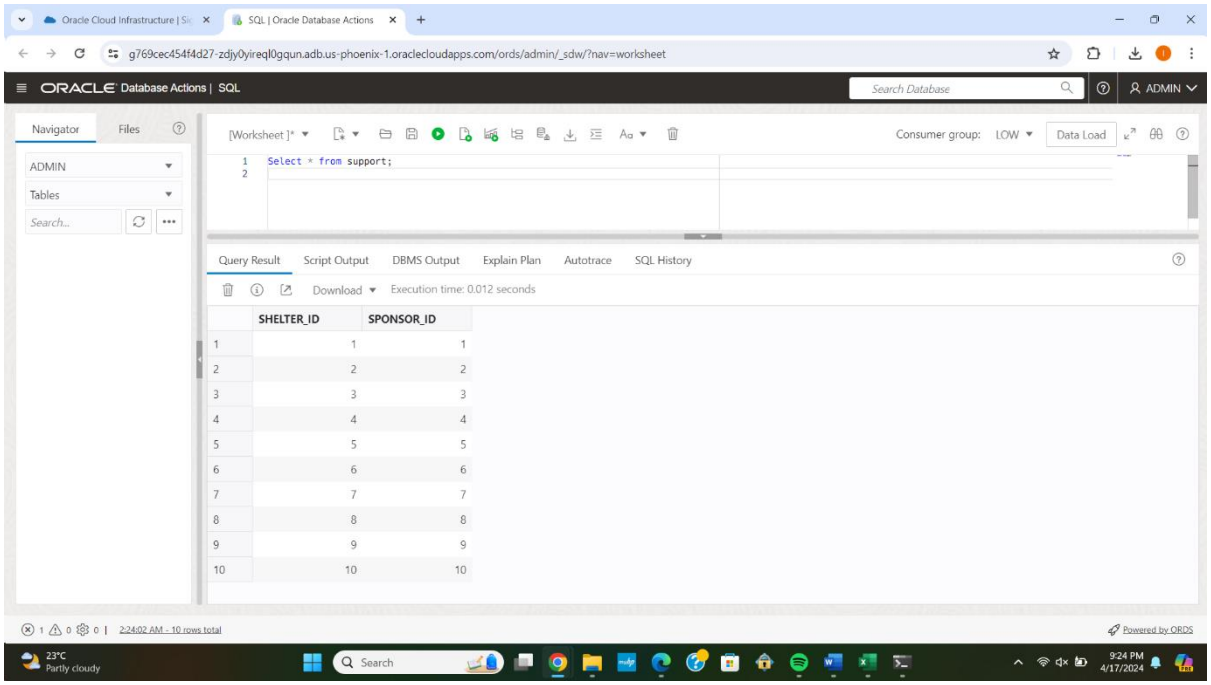


Creating a table named support

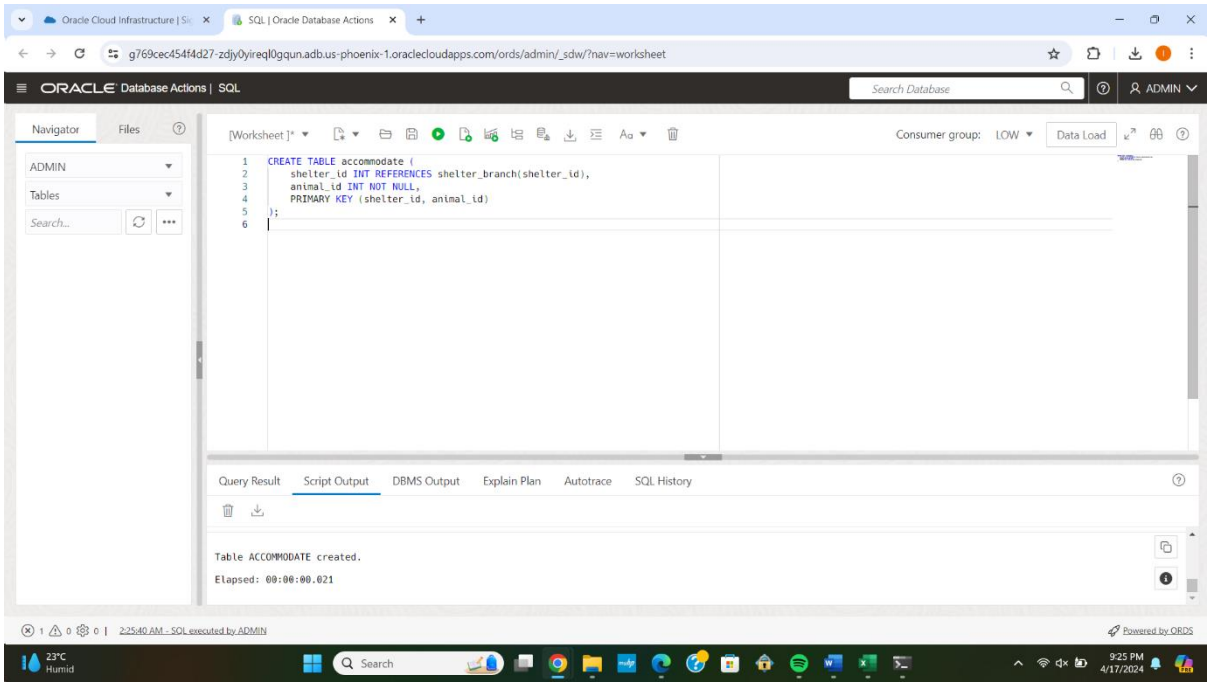


Inserting values to support

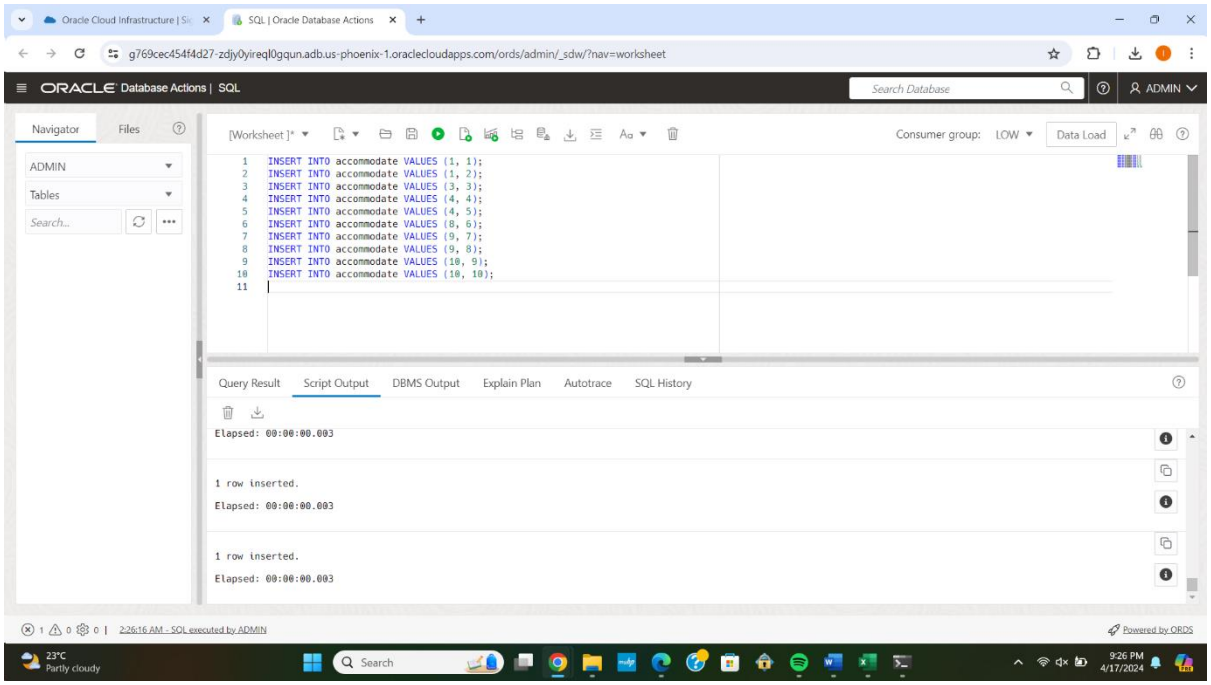


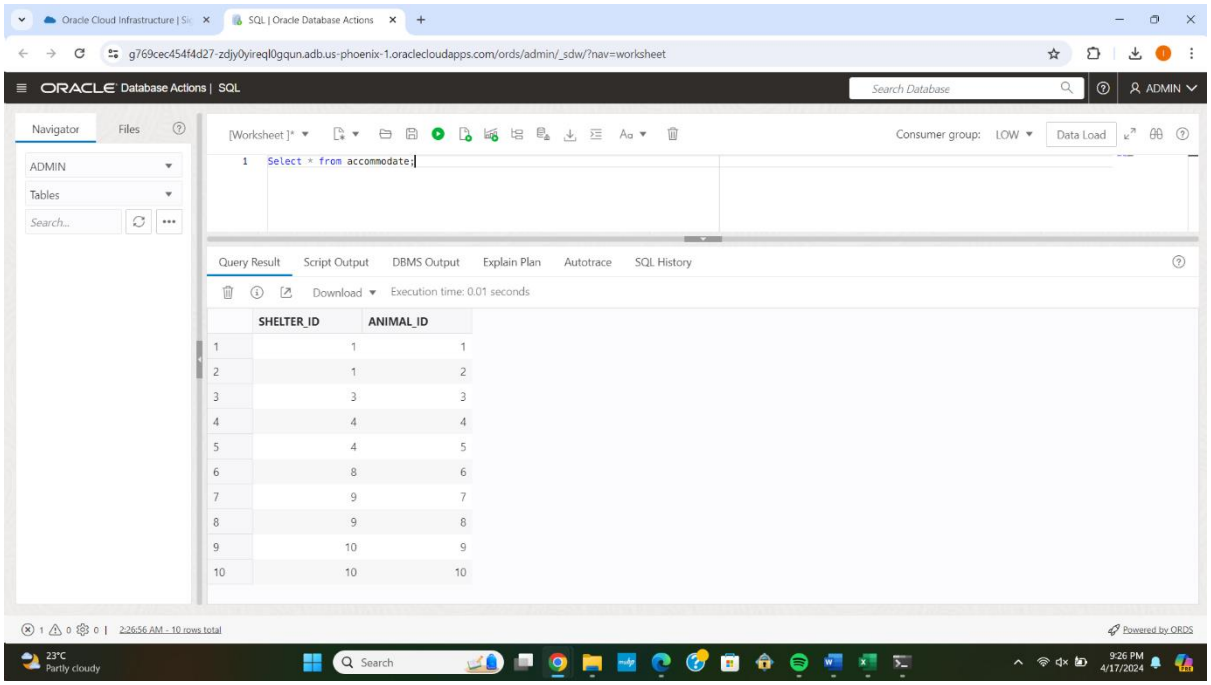


Creating a table named accommodate

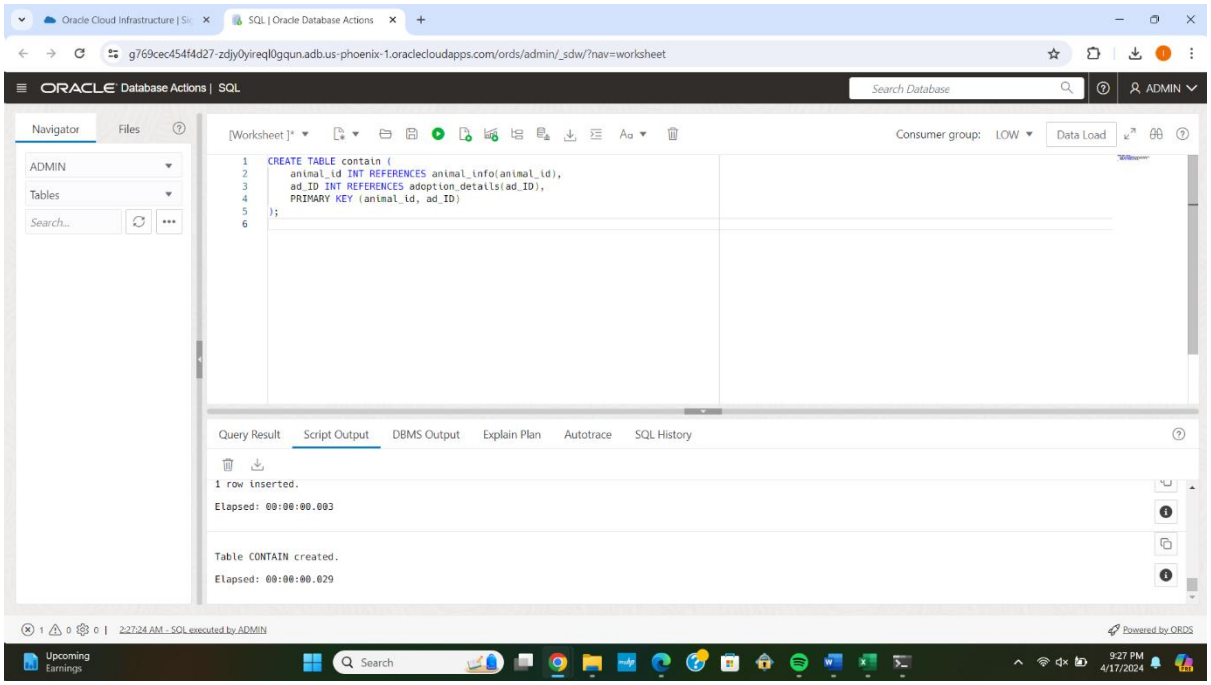


Inserting values to accommodate

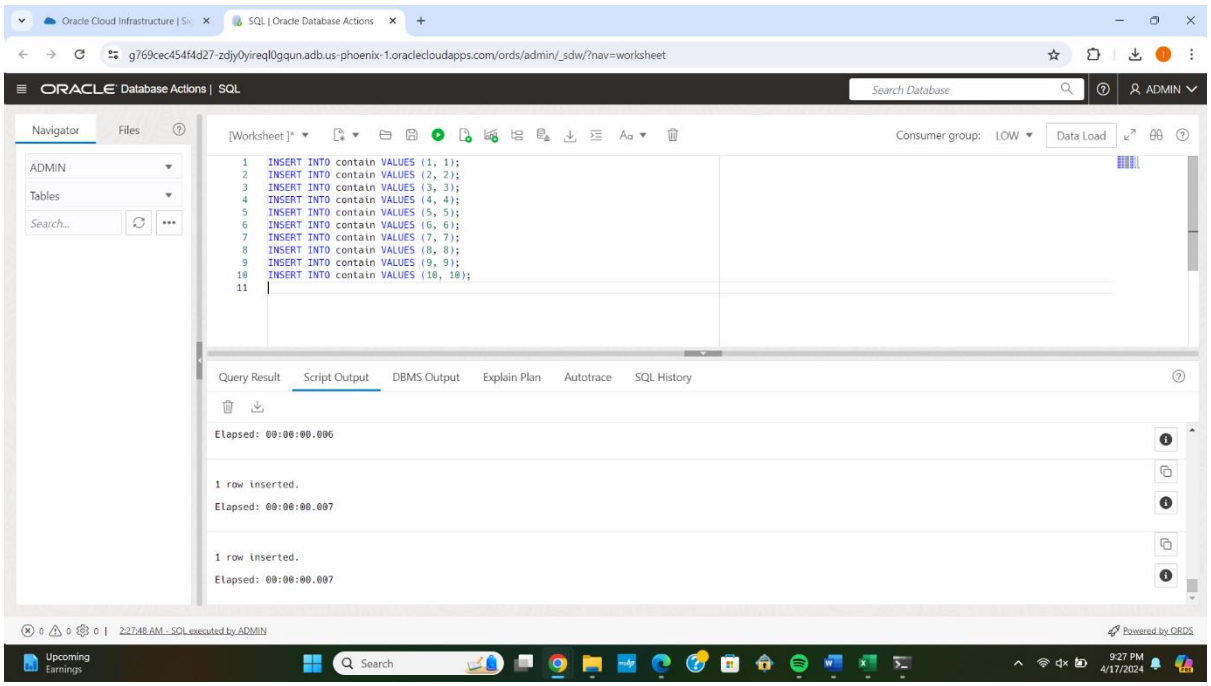


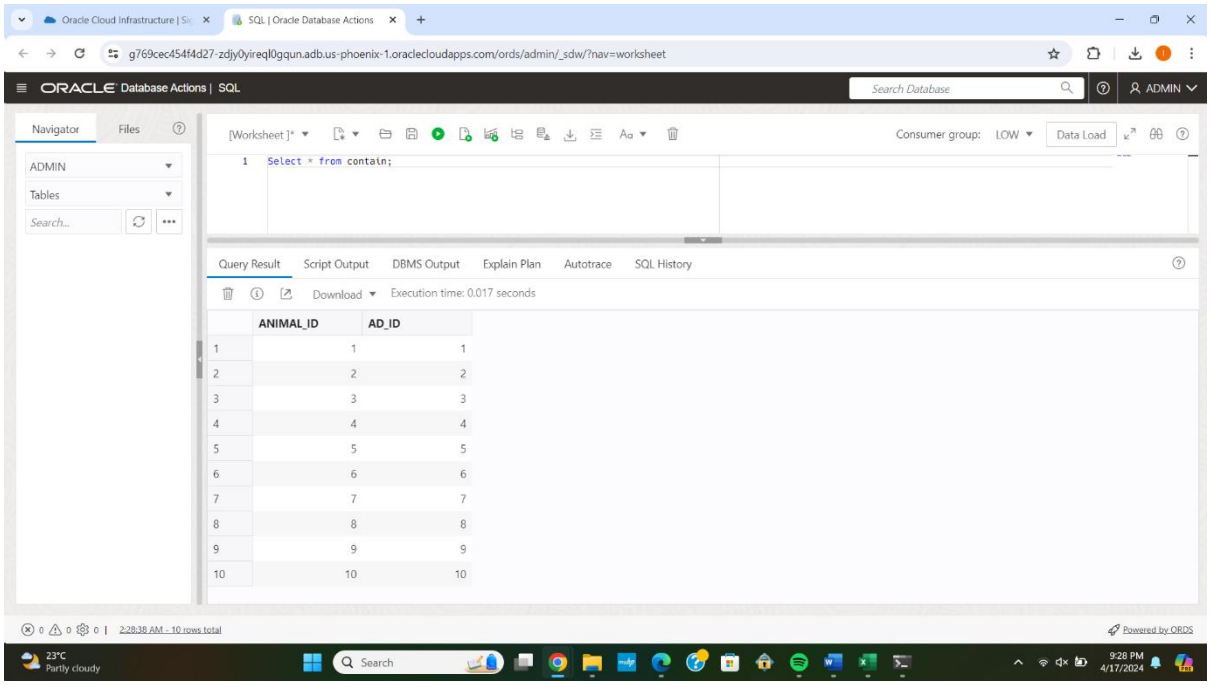


Creating a table named contain

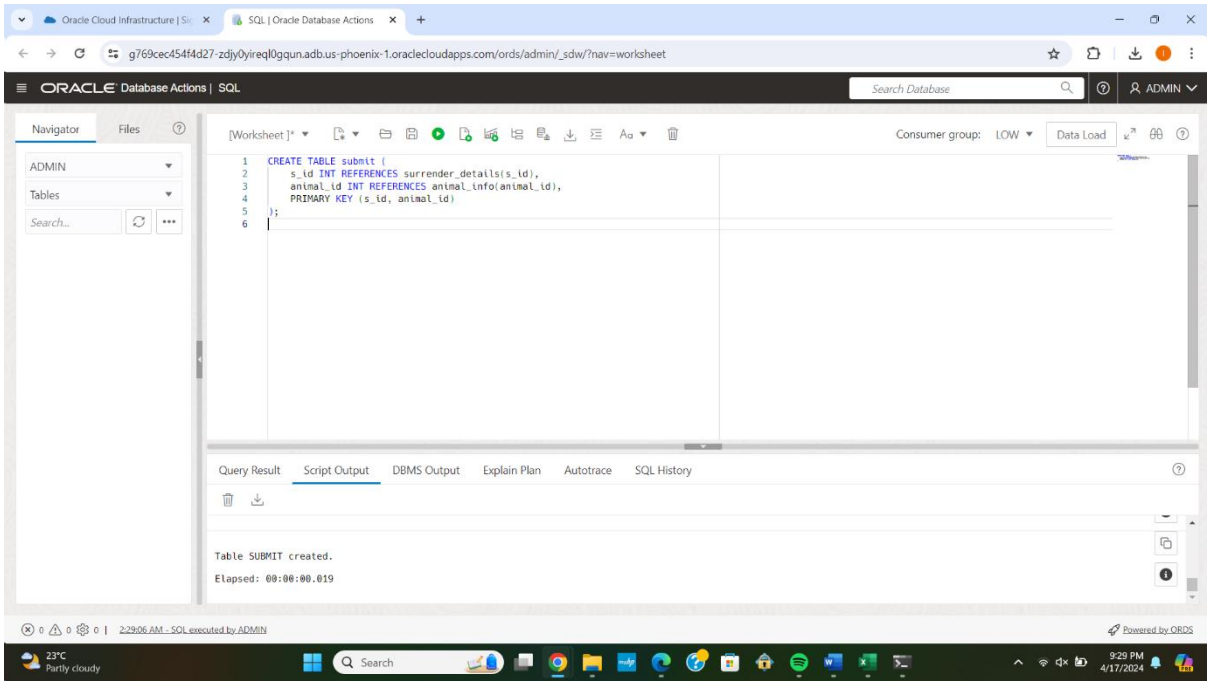


Inserting values to contain

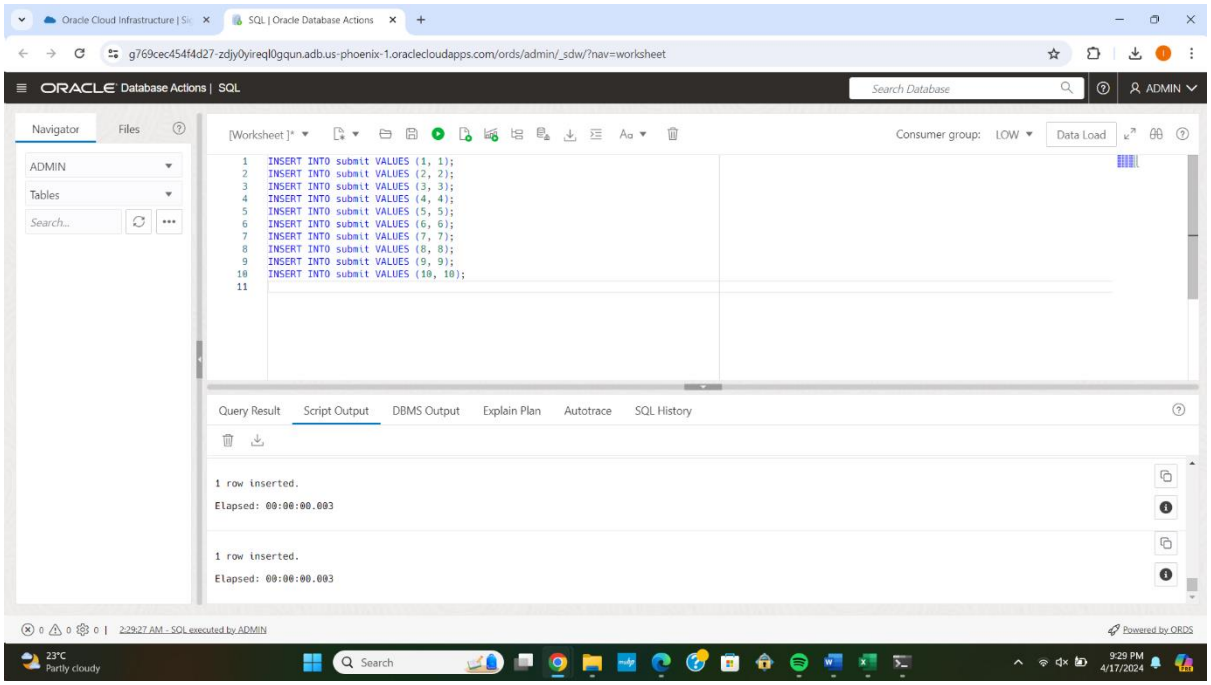


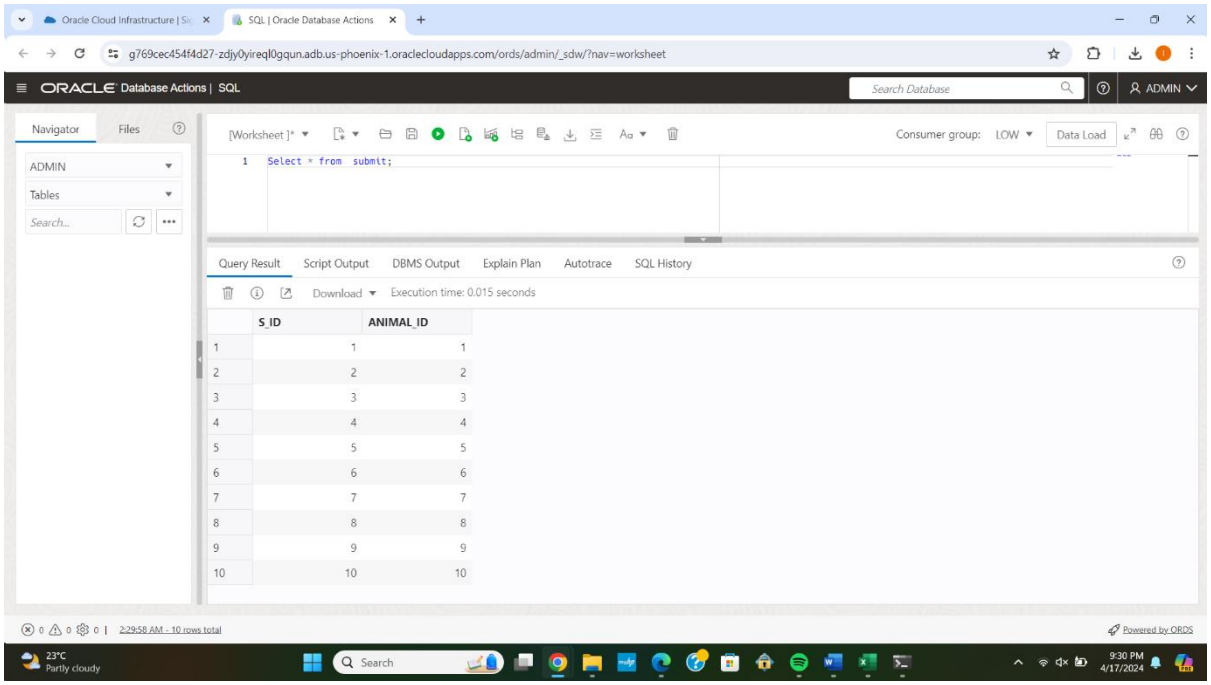


Creating a table named submit

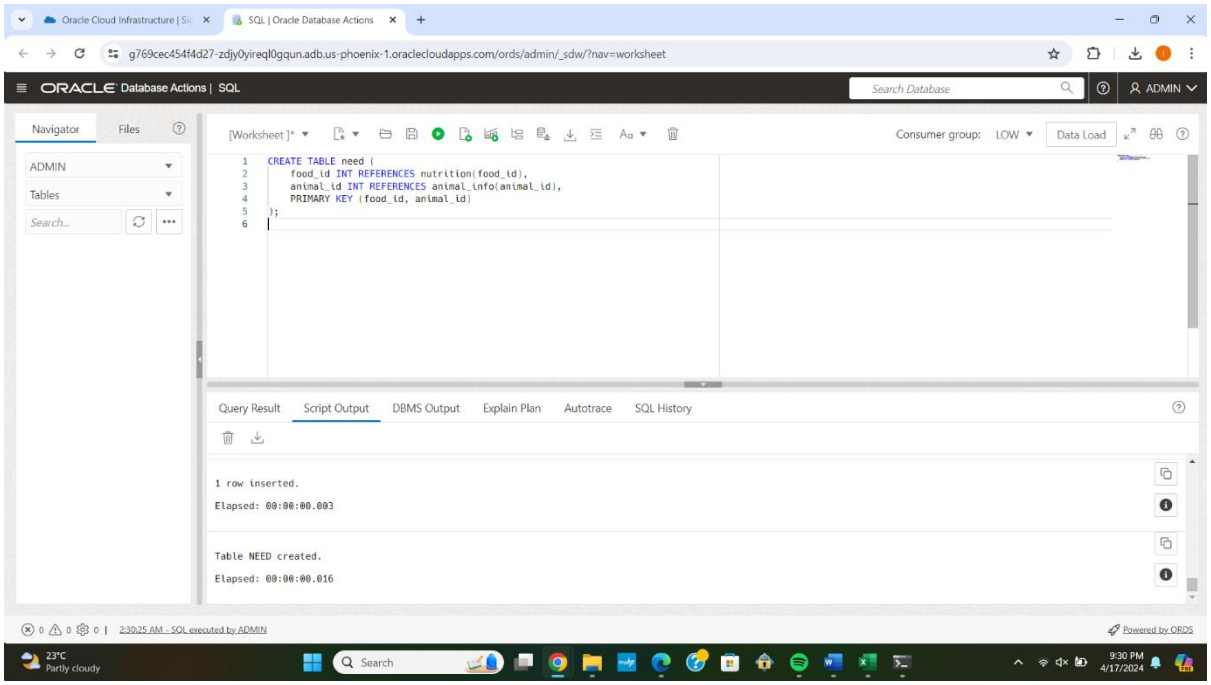


Inserting values to submit

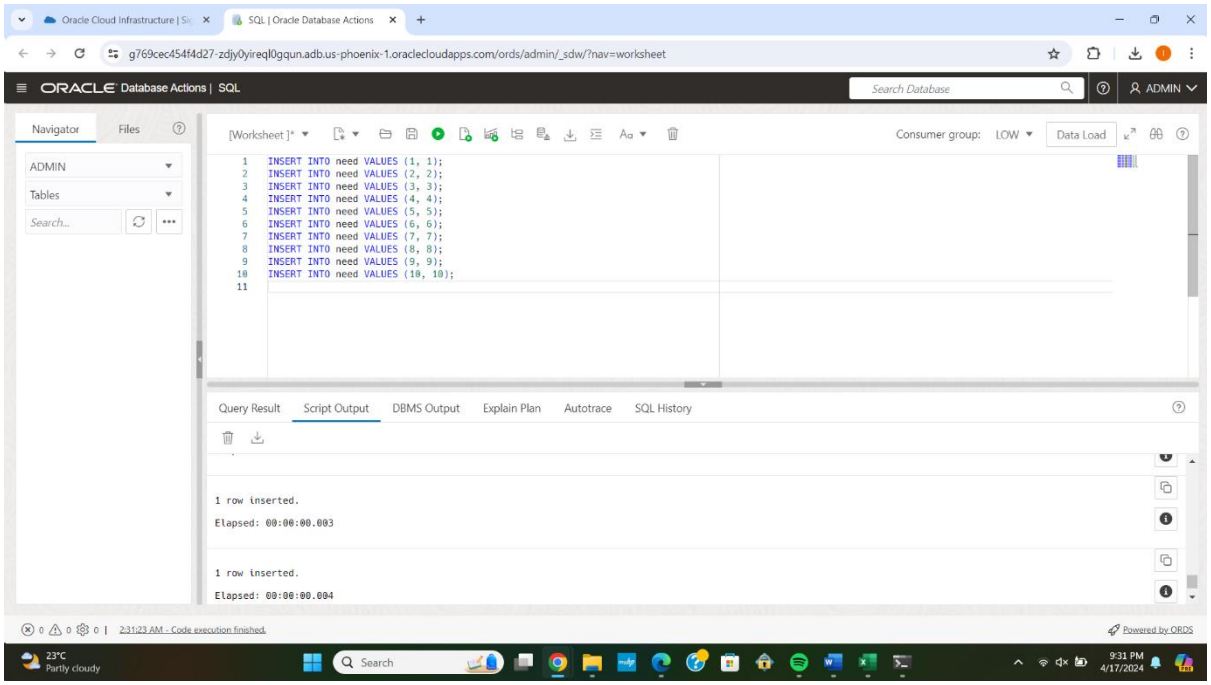


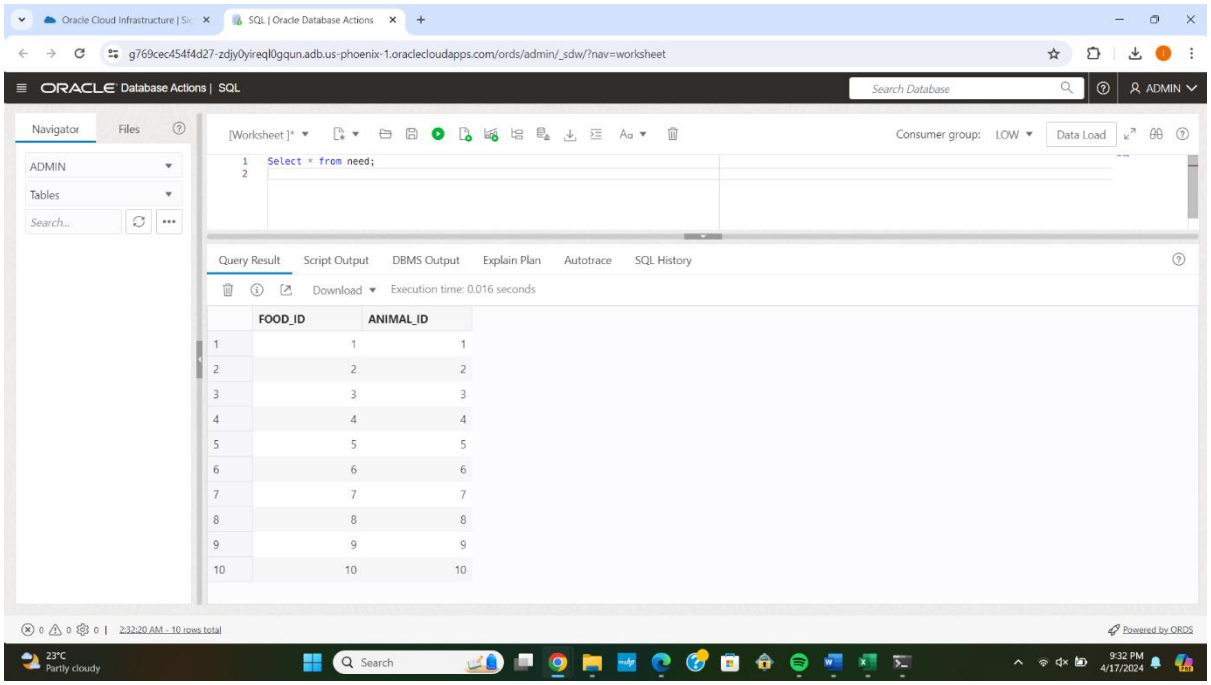


Creating a table named need

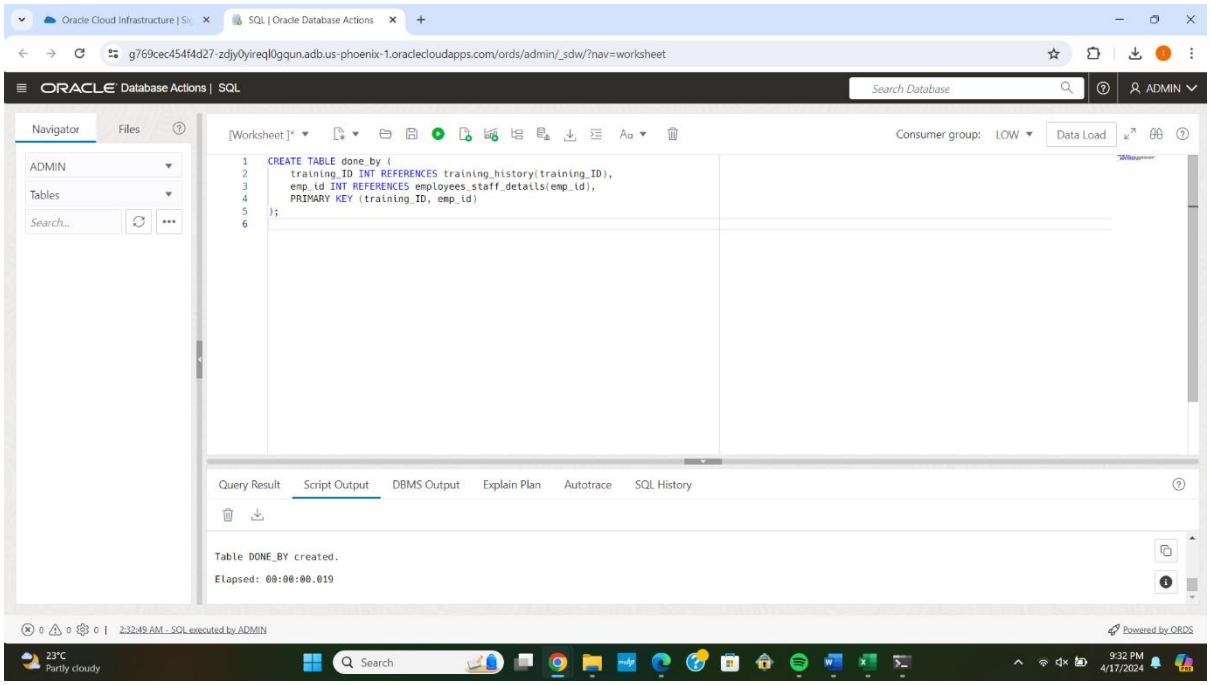


Inserting values to need

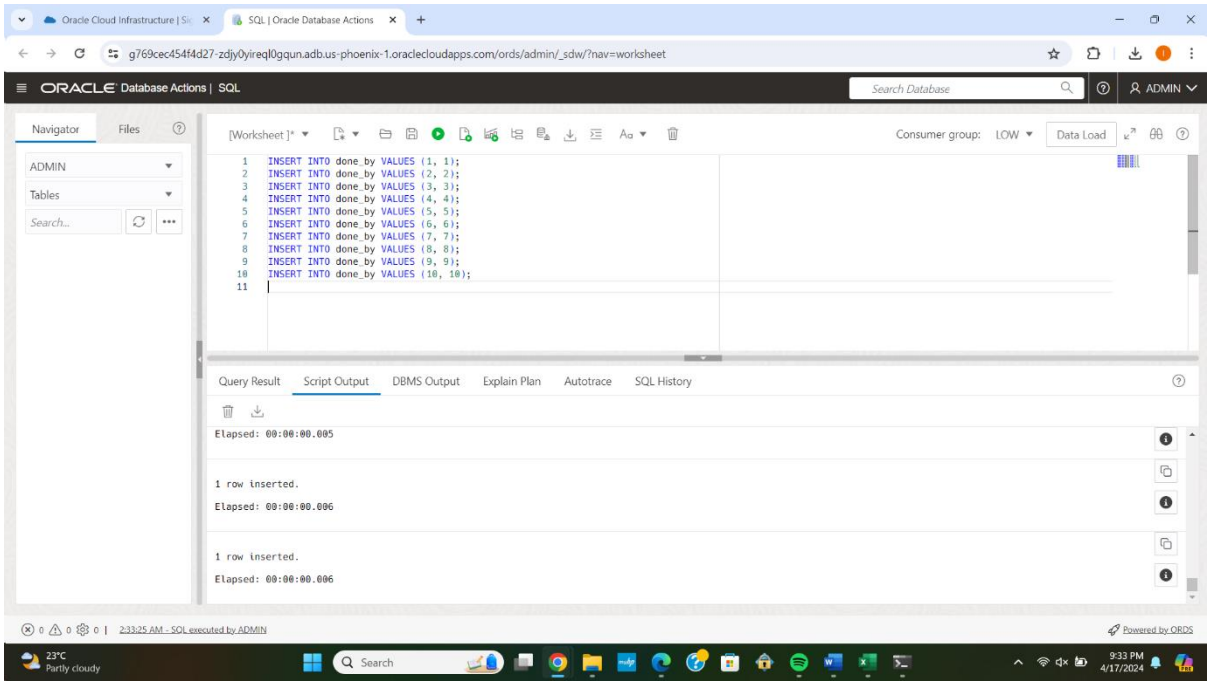


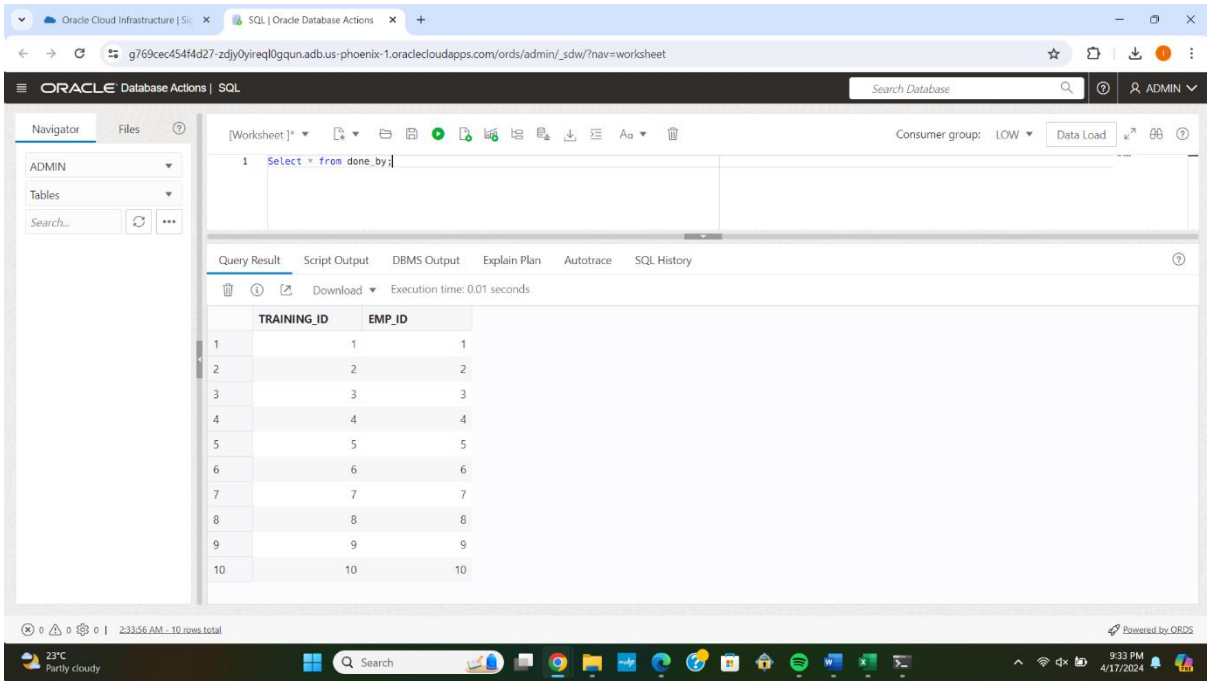


Creating a table named done_by

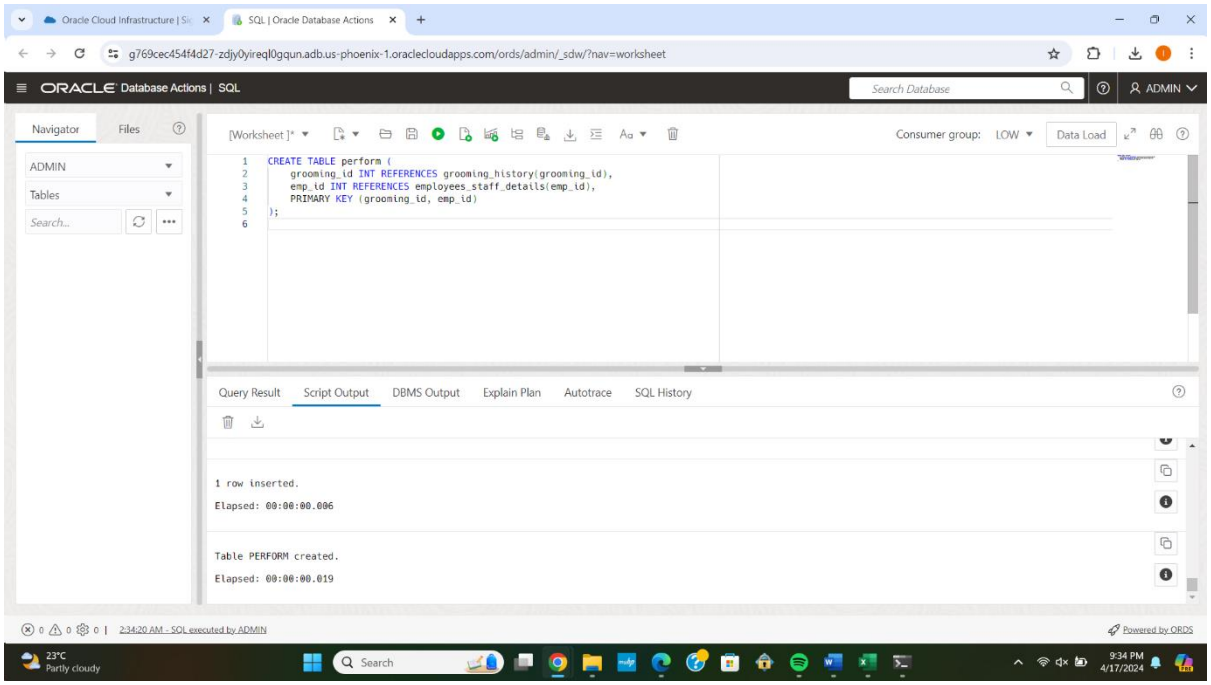


Inserting values to done_by

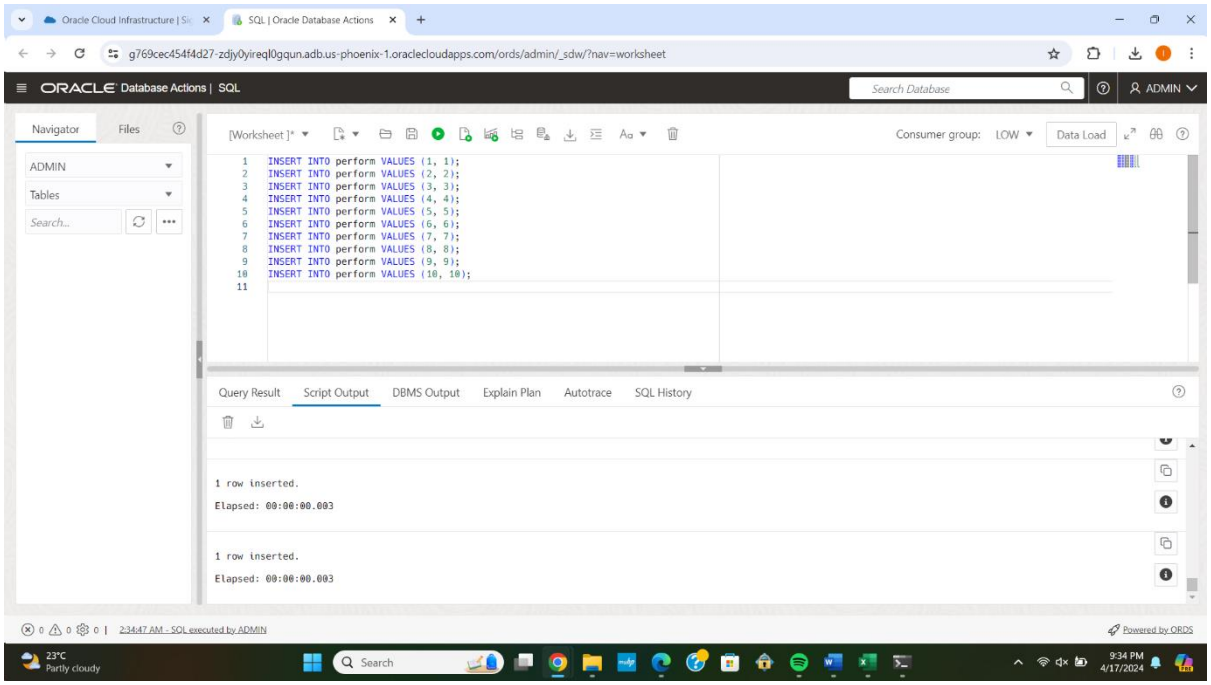


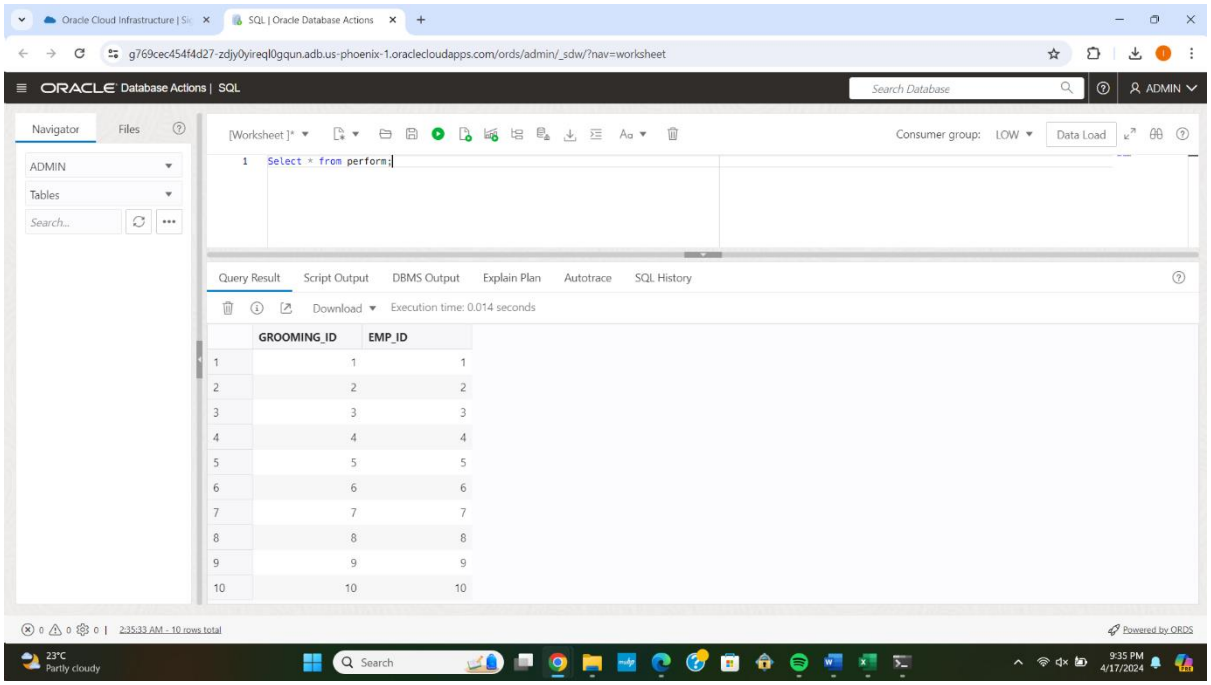


Creating a table named perform

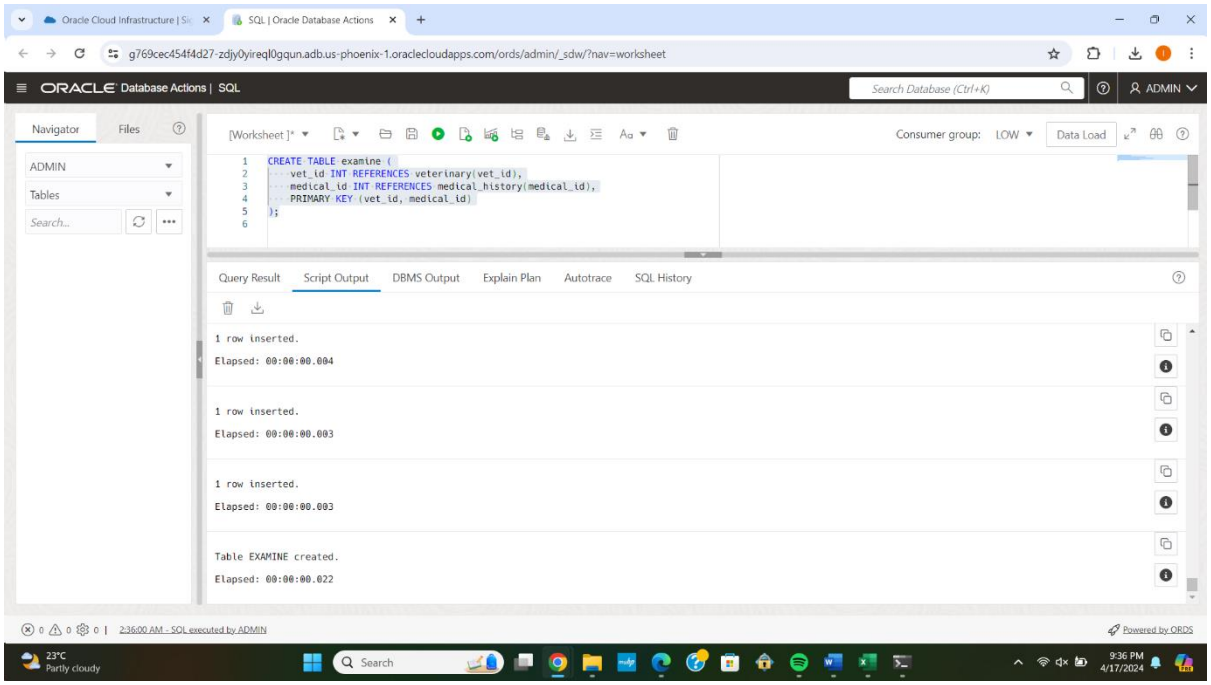


Inserting values to perform

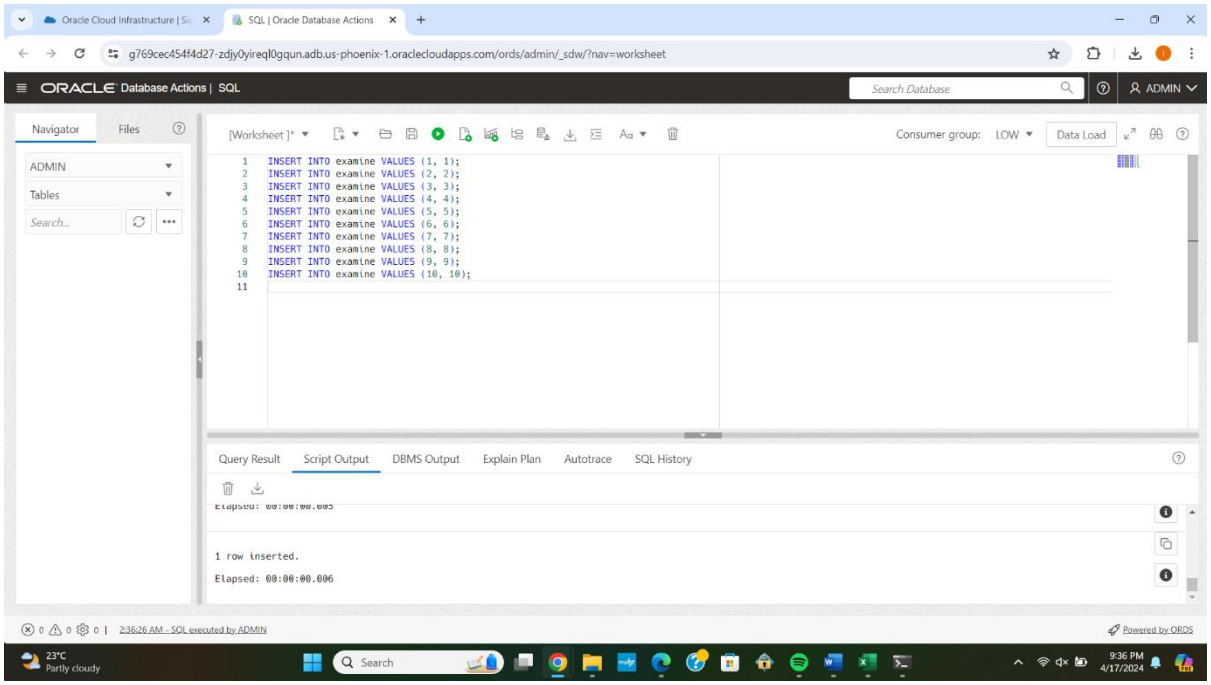


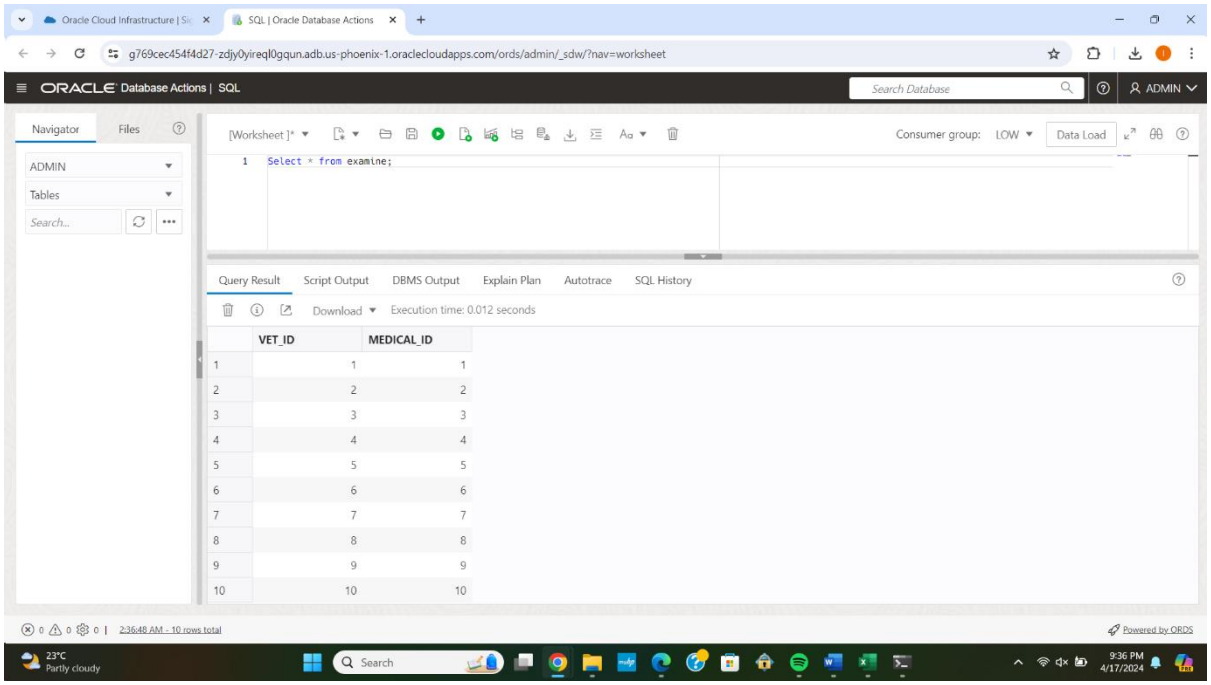


Creating a table named examine

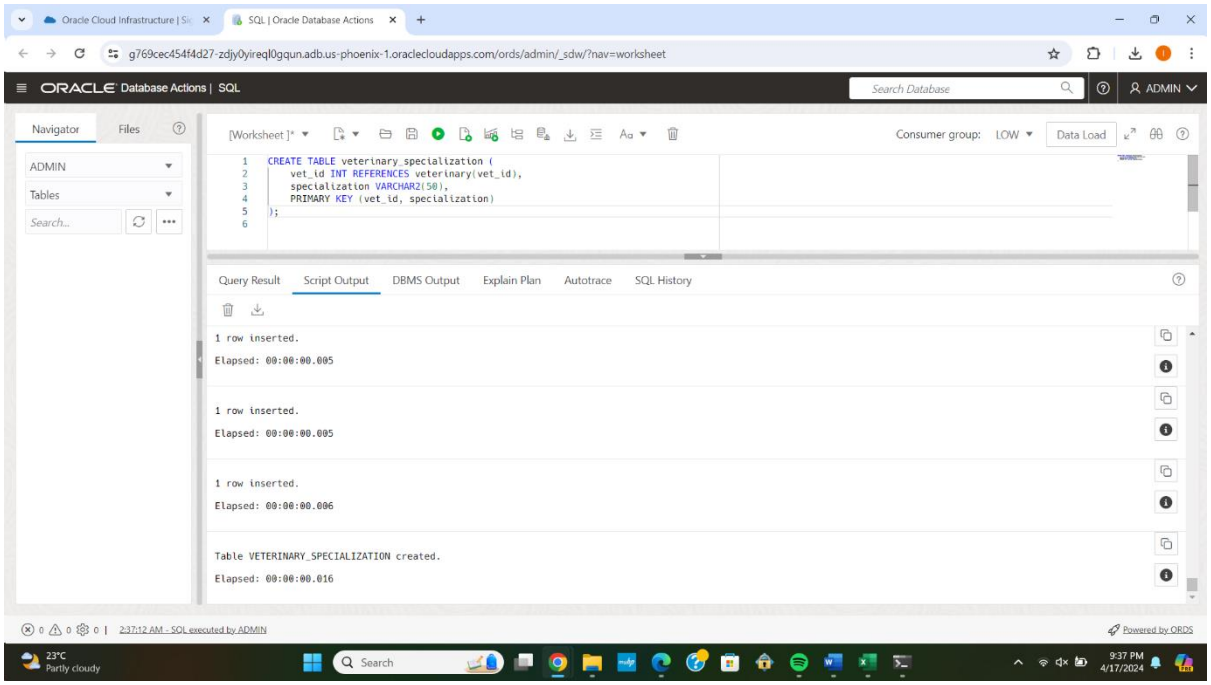


Inserting values to examine

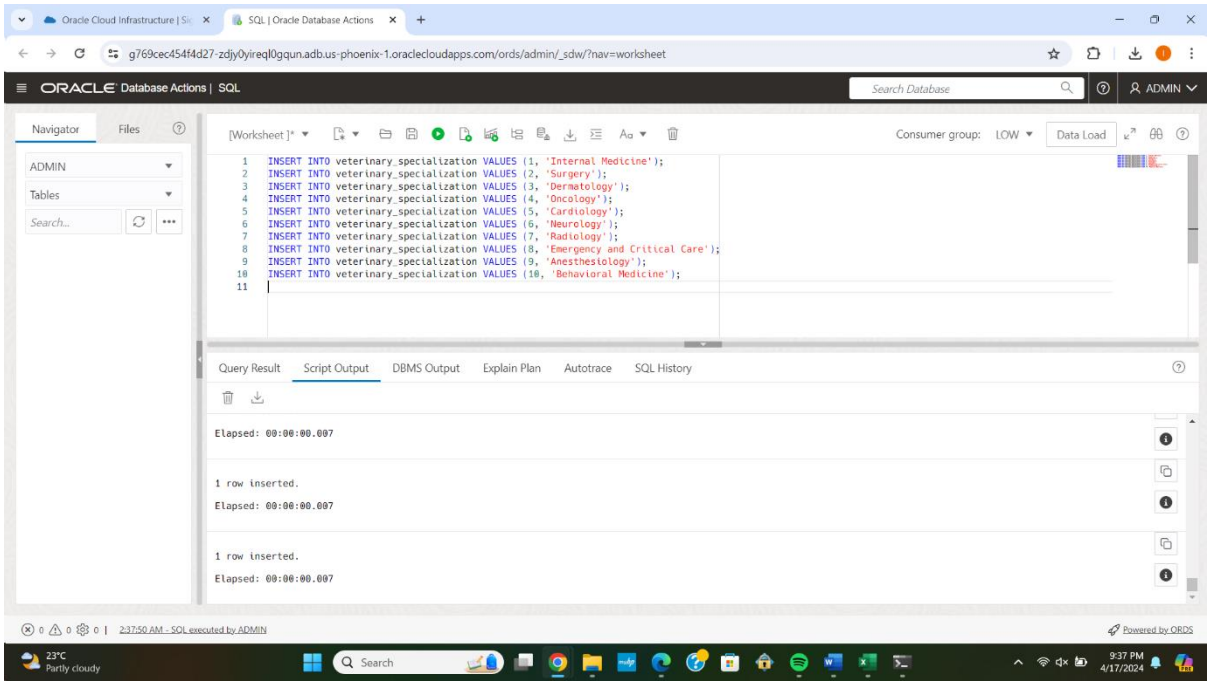


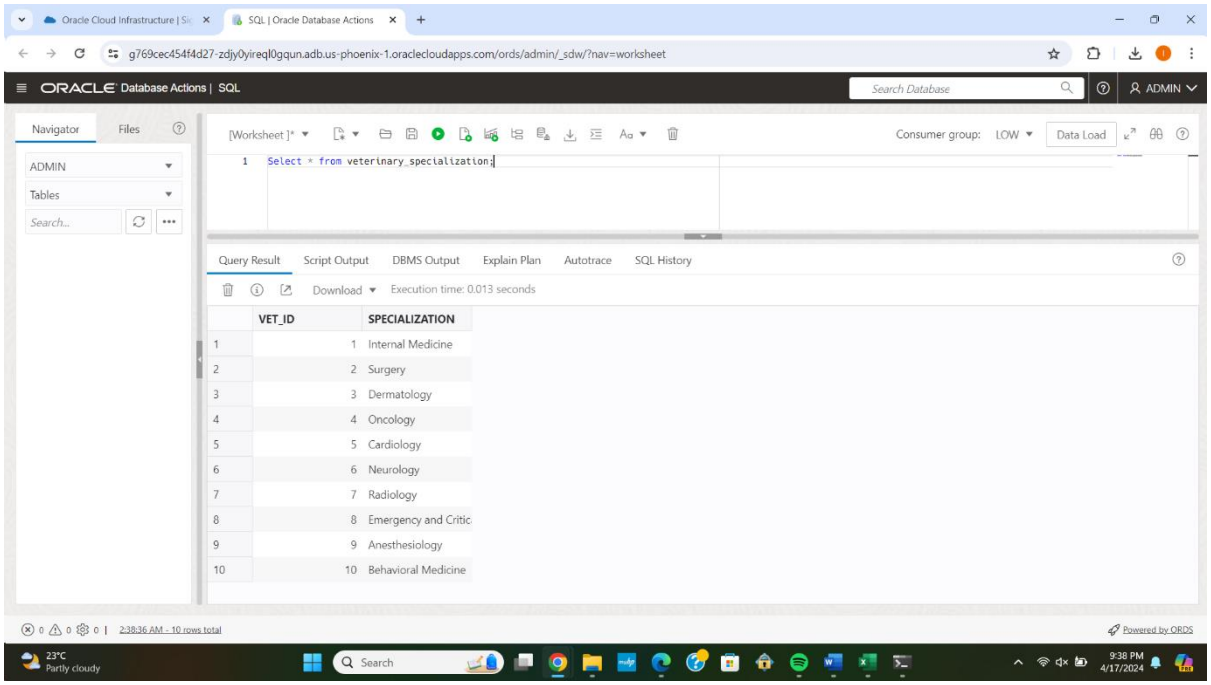


Creating a table named veterinary_specialization

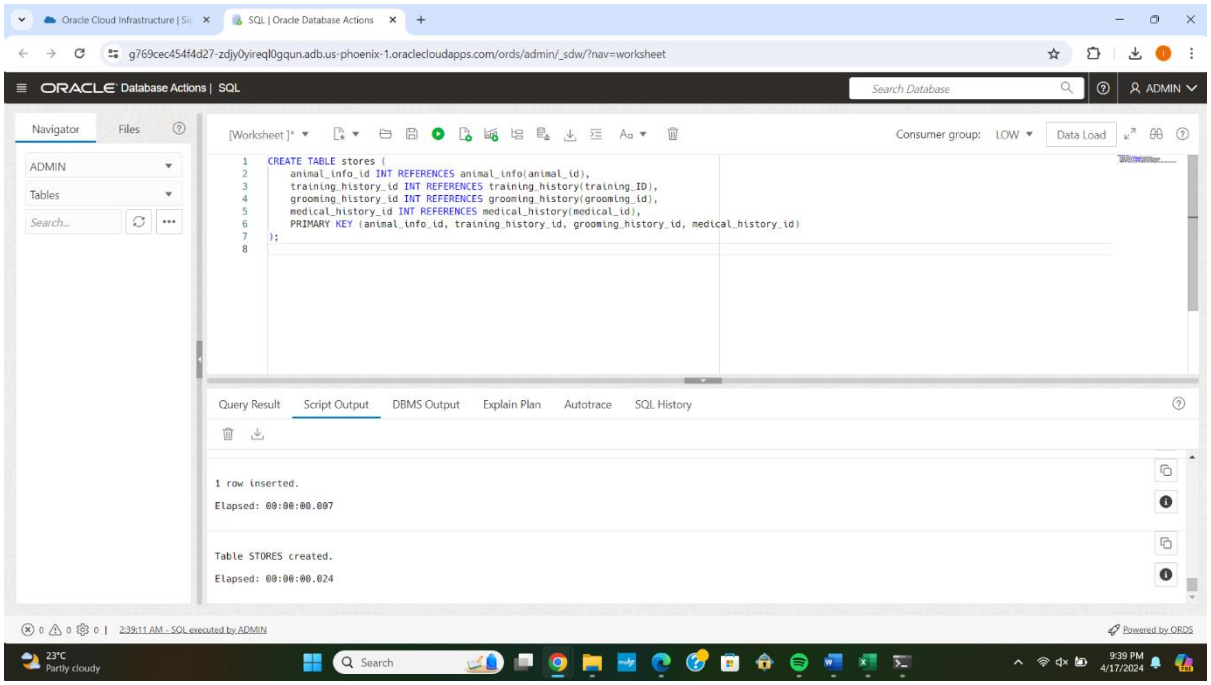


Inserting values to veterinary_specialization

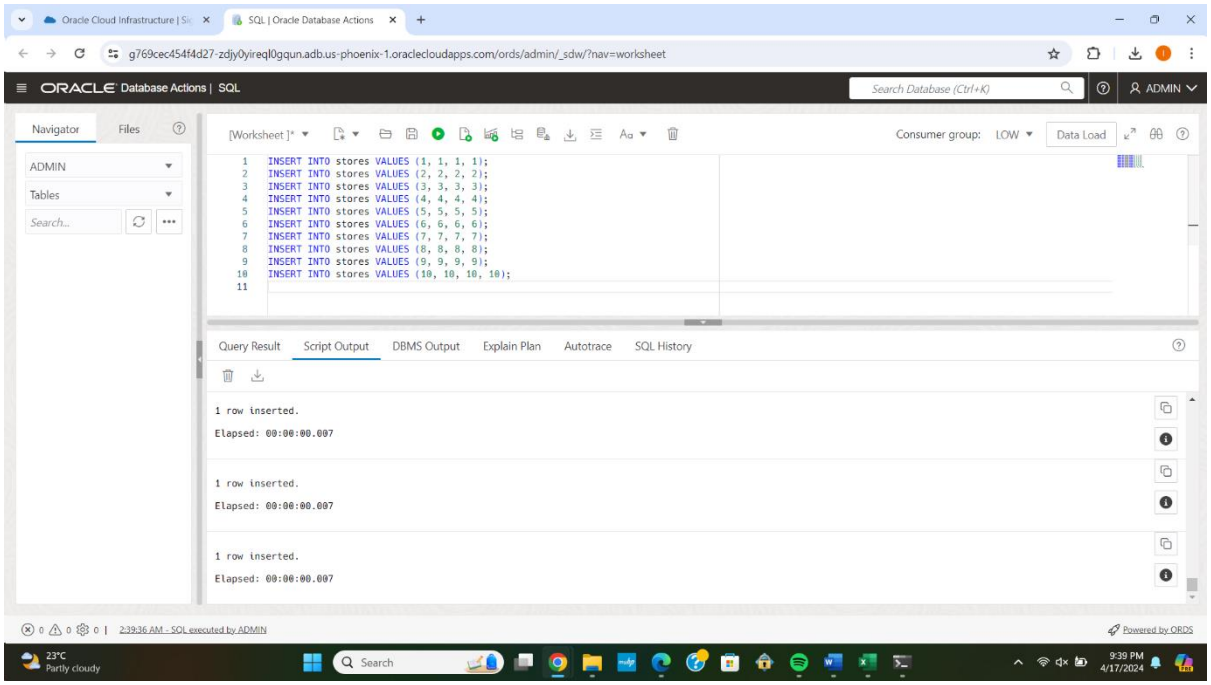


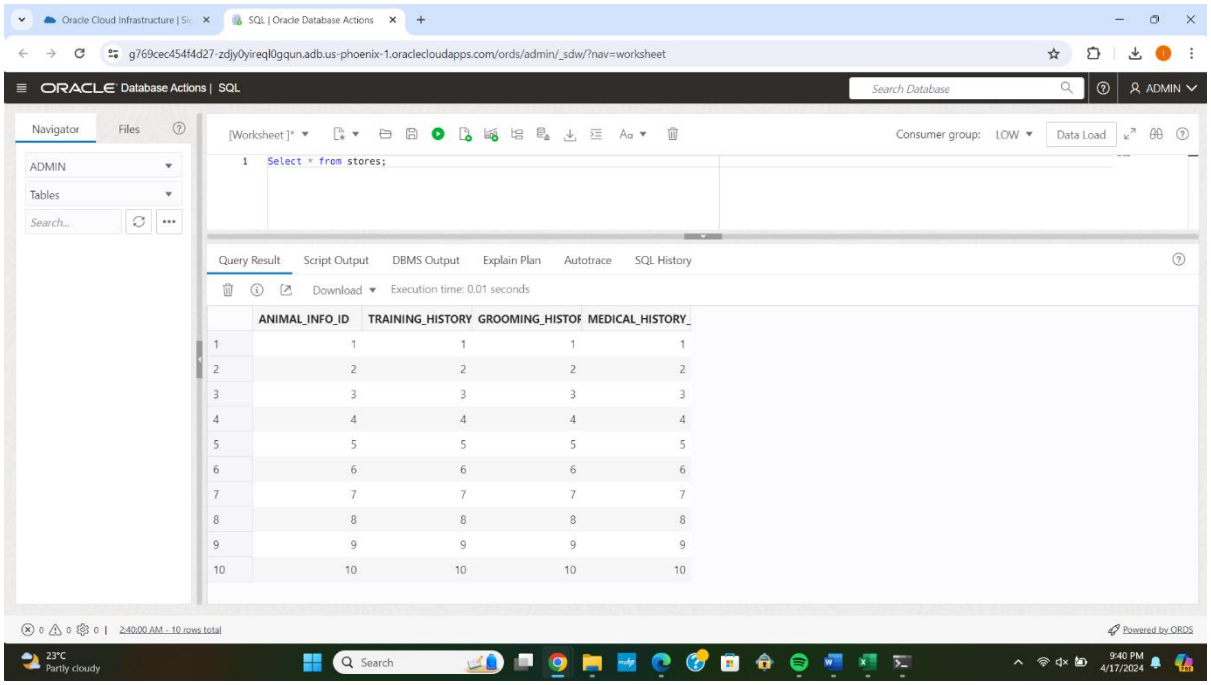


Creating a table named stores



Inserting values to stores


Oracle Cloud Infrastructure | Sc
SQL | Oracle Database Actions
g769cec454f4d27-zdjy0yireql0gqun.adb.us-phoenix-1.oraclecloudapps.com/ords/admin/_sdw/?nav=worksheet
ORACLE Database Actions | SQL
Search Database (Ctrl+K)
ADMIN
Navigator
Files
ADMIN
Tables
Search...
[Worksheet]*
Consumer group: LOW
Data Load
1 INSERT INTO stores VALUES (1, 1, 1, 1);
2 INSERT INTO stores VALUES (2, 2, 2, 2);
3 INSERT INTO stores VALUES (3, 3, 3, 3);
4 INSERT INTO stores VALUES (4, 4, 4, 4);
5 INSERT INTO stores VALUES (5, 5, 5, 5);
6 INSERT INTO stores VALUES (6, 6, 6, 6);
7 INSERT INTO stores VALUES (7, 7, 7, 7);
8 INSERT INTO stores VALUES (8, 8, 8, 8);
9 INSERT INTO stores VALUES (9, 9, 9, 9);
10 INSERT INTO stores VALUES (10, 10, 10, 10);
11
Query Result
Script Output
DBMS Output
Explain Plan
Autotrace
SQL History
1 row inserted.
Elapsed: 00:00:00.007
1 row inserted.
Elapsed: 00:00:00.007
1 row inserted.
Elapsed: 00:00:00.007
2:39:36 AM - SQL executed by ADMIN
Powered by ORDS
23°C
Partly cloudy
Search
9:39 PM
4/17/2024


Oracle Cloud Infrastructure | Sc
SQL | Oracle Database Actions
g769cec454f4d27-zdjy0yireql0gqun.adb.us-phoenix-1.oraclecloudapps.com/ords/admin/_sdw/?nav=worksheet
ORACLE Database Actions | SQL
Search Database
ADMIN
Navigator
Files
ADMIN
Tables
Search...
[Worksheet]*
Consumer group: LOW
Data Load
1 Select * from stores;
Query Result
Script Output
DBMS Output
Explain Plan
Autotrace
SQL History
Download
Execution time: 0.01 seconds
ANIMAL_INFO_ID TRAINING_HISTORY GROOMING_HISTOF MEDICAL_HISTORY_
1 1 1 1 1
2 2 2 2 2
3 3 3 3 3
4 4 4 4 4
5 5 5 5 5
6 6 6 6 6
7 7 7 7 7
8 8 8 8 8
9 9 9 9 9
10 10 10 10 10
2:40:00 AM - 10 rows total
Powered by ORDS
23°C
Partly cloudy
Search
9:40 PM
4/17/2024

INDIVIDUAL CONTRIBUTION

Firstly we started with understanding the functional dependencies and created multiple functional dependencies using the domain knowledge we have. For this we split the total tables into parts and decided to work on 3 to 4 tables each. I started with understand the animal_info tables and the functional dependencies in the table. After understanding that the breed type is different for different animals, I created a functional depends for the same. This indicates that for each animal breed type belongs to a animal type only. For this I found the need to split the table into 2 and mentioned the same to the team. By introducing this FD, we understood the need for dividing the table into 2 and divided it according and also proved that the dependency preservance is maintained and the join is lossless if joined.

Next I analysed the surrender details table and saw that there cannot be another table created or a functional dependency created as the assumptions we have made as per the project will not be in sync with the table if it is splitter into two. The defined functional dependencies were satisfying BCNF and thus nothing we changed. For the employee staff details table, I observed that we can define functional dependencies between the education and the level, department of the employee. As the employee is given a level depending on the education he/she has and also the department aligns with the education. I added two functional dependencies and these were not inline with the BCNF. So, there was a need to create another table to preserve these values. After this we also checked for lossless join and the join was lossless, proving that the BCFN can be done.

Then, on the employee payroll, similar analysis was done and the functional dependencies were identified as the hourly pay of the employee can be determined using the emp_id. So we created a new table for this and checked for lossless join to satisfy BCNF. Apart from this I have also updated the assumptions according and made changes to the SQL code and passed it to the teammates for further execution. I have also analysed and identified the placed where the tables should be added and the schema should be updated. I have analysed the need for showing the proof and the method to follow in order to prove that the join is lossless. Further, I mentioned and shared my knowledge about the super key and lossless joins in order to maintain synchronization.