STAGE 1:

Domain Analysis Report

1.Domain Overview

1.1 Purpose:

To provide a scalable, normalized relational database and full-stack system that records, manages, and analyses data for wildlife conservation and national park operations. The system will capture park administration, biodiversity records, research activities, staff and visitor interactions, incidents and patrols, logistics, permits and funding. It aims to organize scattered information about animals, species, habitats, staff, and visitors into a structured and easily accessible format.

By enabling efficient data storage, monitoring, and reporting, the system supports better conservation planning, enhances operational efficiency, and ensures the protection of wildlife resources.

1.2 Scope:

This project focuses on developing a normalized relational database and full-stack system for managing and analyzing data related to wildlife conservation and national park operations.

The scope of this system includes:

- Creation and maintenance of structured databases for animals, species, habitats, zones, and conservation projects.
- Recording of health checkups, feeding schedules, and habitat details for each animal.
- Management of staff information, including supervisors, caretakers, veterinarians, and drivers.
- Tracking of vehicles, resource stocks, and logistics used in park operations.
- Online registration and tracking of visitors, tickets, and tour packages.
- Issuance and validation of permits for research, safaris, and special activities.

- Recording of conservation projects, including objectives, funding sources, and progress tracking.
- Integration of biodiversity and habitat data to support research and decision-making.
- Implementation of CRUD (Create, Read, Update, Delete) operations for all key entities.
- Web-based interface for accessibility and scalability across departments.

1.3 Target Users

The system is created for a diverse user community involved with wildlife conservation, park management, and research and will primarily target the following:

- **Park Managers** – For day-to-day work in managing the park and people, the various zones of the park, logistics, and drawing management reports.
- **Veterinarians** – Who will enter and track animal health data, treatments, and medical histories.
- **Caretakers / Rangers** – For keeping records of feeding animals, maintaining habitats, and daily monitoring appraisals.
- **Maintenance Staff / Logistics** – To track and maintain vehicles, equipment, and resource stocks.
- **Researchers / Biologists** – To access biodiversity and conservation data for analysis and studies.
- **Supervisors** – To approve permits, oversee conservation projects, and manage staff performance.
- **Visitors / Tourists** – To register for visits, book tickets or tour packages, and access basic park information.
- **Funding Agencies / Conservation Authorities** – To review reports on conservation projects, funding utilization, and outcomes.

2. Business Rules

Species Conservation & Ecological Integrity Rules

1. Species classified under *Endangered* or *Vulnerable* categories **must have conservation protocols assigned** in the system.

2. No species record may be modified or deactivated without **biologist-level authorization**.

3. Breeding, mortality, and migration events **must be logged as irreversible records**, with reasons documented and verified by supervisory staff.

Staff Authorization & Operational Discipline Rules

1. Every staff member must have a clearance level and the system must **deny access to features or zones beyond their clearance**.

2. Tasks such as animal feeding, habitat inspections, patrol duties, and medical activities **must be logged and digitally verified**. Failure to perform a scheduled duty (feeding, inspection, patrol, medical checks) **must trigger an automatic escalation to the zone supervisor**.

3. Armed patrols, poaching incidents, or sensitive operations require **multi-level authorization** and cannot be executed without digital approval.

Visitor Regulation & Permit Enforcement Rules

1. Entry into the national park requires a valid permit; **no visitor record may exist without at least one associated permit**.

2. Permit validity (date, time, zone access) must be strictly enforced; the system must **deny zone access if the permit does not authorize it**.

3. Visitors entering Ecologically Sensitive Zones (ESZ) **must be accompanied by an authorized guide**, and this association must be recorded.

Research & Conservation Activity Governance Rules

1. No research activity may commence without a registered project ID and **an approved research permit** linked to the researcher.

2. Projects involving biological sampling, tagging, collaring, or animal sedation **require pre-approval from conservation authorities** and must follow regulatory guidelines.

3. Project conclusions, findings, and datasets must be archived; deletion requires top-level administrative approval.

Medical, Emergency, and Incident Response Rules

1. Every medical interaction with an animal (check-up, vaccination, treatment, sedation) **must be recorded immediately after completion**. Only licensed veterinarians may administer sedatives, tranquilizers, or controlled substances; each use must be tracked with dosage and purpose.

2. Emergencies involving animal injury, human-wildlife conflict, disease outbreaks, or unusual behavior **must be registered as priority incidents**. If an emergency is logged and no response action is taken within the defined window, the system **must escalate to higher authorities**.

3. Mortality events must be recorded with confirmed cause-of-death documentation, verified by a veterinarian or supervisor.

Inventory, Weapons, and Restricted Material Compliance Rules

1. All critical supplies (feed, medical stock, fuel, tranquilizers, weapons, field equipment) **must maintain minimum threshold levels**, and shortages must generate automatic restock alerts.

2. Restricted materials such as tranquilizers or firearms **require dual authorization** before checkout and must be returned with usage logs.

3. Inventory entries cannot be deleted; corrections require supervisor approval and must leave a traceable audit record.

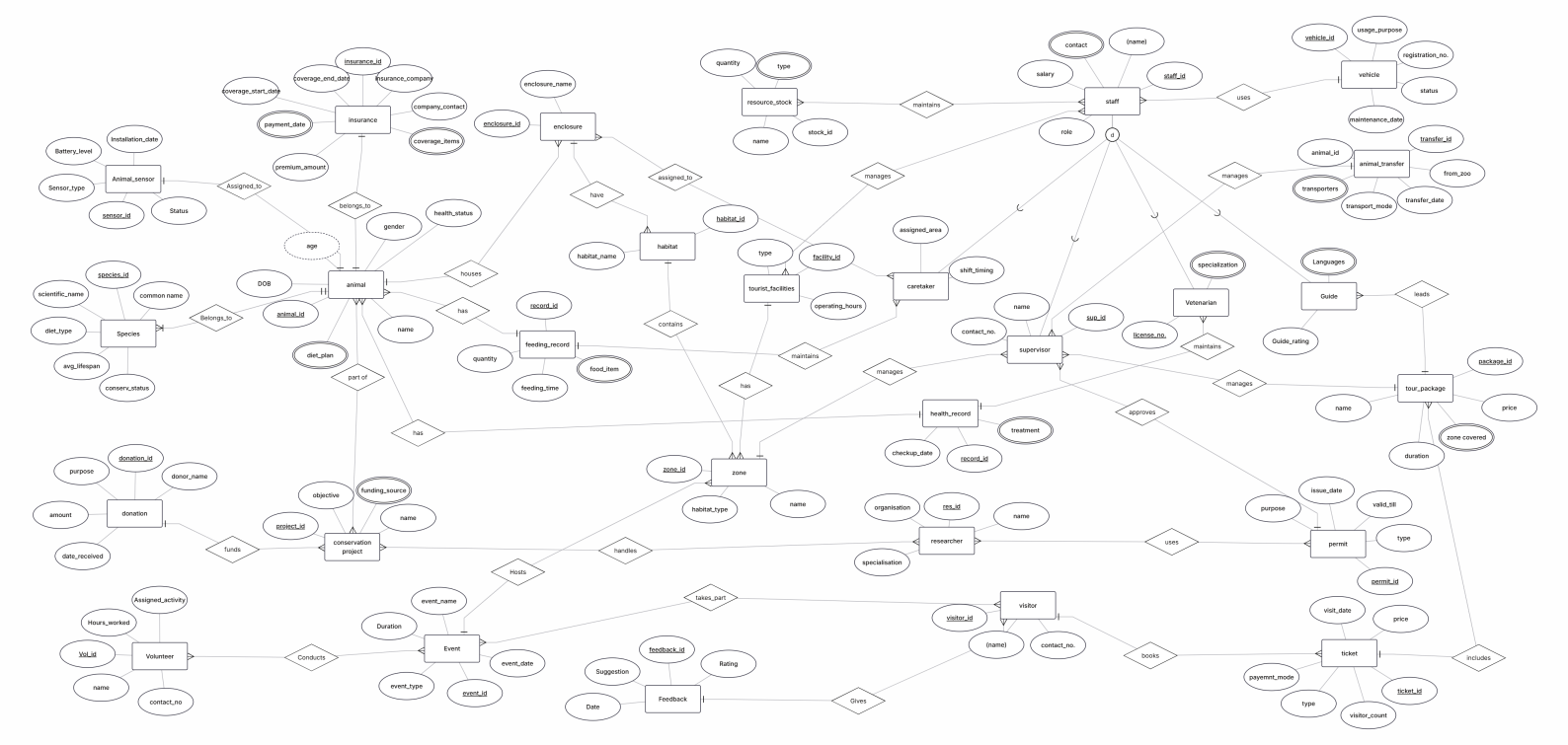
4. Vehicles must be linked to responsible staff and cannot be checked out without digital authorization.

3. Preliminary list of entities

- Animal – Represents each individual animal in the park
- Species – Captures classification information for animals: e.g. a scientific name, common name, conservation status (such as endangered, vulnerable), and species-specific notes (lifespan, behaviour etc.)
- Habitat – Represents the natural environment where animals live, including habitat type, area, and zone.
- Zone – Defines different areas of the national park; each zone can contain multiple habitats. Defines a broader section of the park.
- Staff – Records all park employees; ID, full name, role, contact information and others.
- Health Record – Keeps records of animal medical check-ups, treatments, vaccinations, and veterinarian details.
- Feeding Record – This logs feeding activities like feeding time, food type, quantity, and caretaker responsible.
- Vehicle – Maintains data about park vehicles used in park operations.
- Visitor – Holds data about individuals who visit the park
- Ticket – Contains ticket booking information including visitor details, date etc.
- Permit – Records permit details for research or special access.
- Conservation Project – Stores data about conservation projects, their objectives, funding sources etc.

4. Possible System Outputs and Use-cases

- Animal Record Management – Add, update, view animal details including species, habitat, and health status.
- Health and Treatment Reports – Generate reports of medical check-ups, treatments, and vaccinations for each animal.
- Feeding Schedules – Display and print daily and weekly feeding logs for all animals.
- Habitat and Zone Monitoring – Track details of each zone, its habitats, and the animals living there.
- Staff Management – Maintain staff profiles, assign roles, and record responsibilities.
- Visitor and Ticket Management – Register visitors, book tickets, and generate visit summaries.
- Permit Issuance and Approval – Manage and track permits for research, filming, or special park access.


insurance_id
coverage_end_date
insurance_company
coverage_start_date
company_contact
insurance
payment_date
coverage_items
premium_amount
Installation_date
Battery_level
Sensor_type
Animal_sensor
sensor_id
Status
Assigned_to
belongs_to
age
gender
health_status
DOB
animal
houses
has
species_id
scientific_name
common name
diet_type
Species
avg_lifespan
conserv_status
Belongs_to
animal_id
name
diet_plan
part of
enclosure_name
enclosure_id
enclosure
have
assigned_to
habitat_id
habitat_name
habitat
contains
record_id
quantity
feeding_record
feeding_time
food_item
maintains
has
quantity
type
resource_stock
name
stock_id
maintains
contact
(name)
salary
staff_id
staff
role
d
uses
vehicle_id
usage_purpose
registration_no.
vehicle
status
maintenance_date
manages
animal_id
transfer_id
animal_transfer
from_zoo
transporters
transport_mode
transfer_date
manages
assigned_area
shift_timing
caretaker
manages
type
facility_id
tourist_facilities
operating_hours
has
specialization
Vetenarian
license_no.
maintains
Languages
Guide
Guide_rating
leads
name
contact_no.
sup_id
supervisor
approves
manages
package_id
tour_package
name
price
zone covered
duration
has
health_record
treatment
checkup_date
record_id
zone_id
zone
habitat_type
name
handles
organisation
res_id
name
researcher
specialisation
uses
issue_date
purpose
valid_till
permit
type
permit_id
donation_id
purpose
donor_name
amount
donation
date_received
funds
objective
funding_source
project_id
name
conservation project
Hosts
Assigned_activity
Hours_worked
Vol_id
Volunteer
name
contact_no
Conducts
event_name
Duration
Event
event_date
event_type
event_id
takes_part
visitor_id
(name)
contact_no.
visitor
books
visit_date
price
ticket
payemnt_mode
type
ticket_id
visitor_count
includes
feedback_id
Suggestion
Rating
Date
Feedback
Gives

STAGE:2
ER Diagram and Conceptual Design Report

ENTITY AND ATTRIBUTES

No.	Entity Name	Primary Key (PK)	Key / Important Attributes	Purpose / Description
1	**Zone**	zone_id	name, habitat_type	Divides the national park into manageable areas.
2	**Habitat**	habitat_id	habitat_name	Represents natural environments that contain animals and plants.
3	**Animal**	animal_id	name, gender, age, diet_plan, health_status, DOB	Stores details of animals in the park.
4	**Conservation_Project**	project_id	name, objective, funding_source	Tracks ongoing or completed wildlife conservation projects.
5	**Researcher**	res_id	name, organisation, specialisation	Represents individuals conducting research within the park.
6	**Permit**	permit_id	purpose, issue_date, valid_till, type	Records permits granted to researchers or staff for official activities.

7	**Staff**	staff_id	name, contact, salary, role	Parent entity for specialized staff roles (Guide, Caretaker, Veterinarian).
8	**Guide**	staff_id	languages, guide_rating	Specialized staff responsible for tours and visitors.
9	**Caretaker**	staff_id	assigned_area, shift_timing	Specialized staff responsible for animal care and feeding.
10	**Veterinarian**	staff_id	license_no, specialization	Specialized staff responsible for animal health and treatment.
11	**Supervisor**	staff_id	name, contact_no	Manages and supervises park operations and staff.
12	**Visitor**	visitor_id	name, contact_no	Represents tourists or park visitors.
13	**Ticket**	ticket_id	visit_date, payment_mode, type, visitor_count	Records details of park entry or tour bookings.
14	**Tour_Package**	package_id	name, duration, zone_covered, price	Defines available tour packages and safaris.
15	**Vehicle**	vehicle_id	registration_no, usage_purpose, maintenance_date, status	Represents vehicles used for tours or conservation work.

16	**Health_Record**	record_id	checkup_date, treatment,	Tracks animal health checkups and treatments by vets.
17	**Feeding_Record**	record_id	feeding_time, quantity, food_item	Logs feeding activities handled by caretakers.
18	**Resource_Stock**	stock_id	type, quantity	Tracks available resources (e.g., food, equipment) for maintenance.
19	**Animal_Sensor**	sensor_id	Battery_level, sensor type	Stores details of sensors attached to animals for tracking and monitoring.
20	**Event**	event_id	Event type, event_date, duration	Stores details of events conducted in the park, such as workshops, guided tours, festivals, and training programs.
21	**Volunteer**	volunteer_id	Name, contact_no,	Contains information about volunteers who assist with events, conservation work, visitor support, and other park operations.
22	**Species**	species_id	Scientific name, conserv_status,	A master list of all species in the park, including scientific classification, habitat type, and conservation status.

23	**Feedback**	feedback_id	rating, suggestion, date	Stores visitor feedback and suggestions about park experience.
24	**Tourist Facilities**	facility_id	Type, operating hours	Stores details of visitor amenities available in the park.
25	**Donations**	donation_id	Donor name, amount, purpose	Records contributions made by visitors or sponsors, including amount, date, and purpose of the donation.
26	**Enclosure**	enclosure_id	enclosure_name	Physically bounded area inside the national park, designed based on habitat needs,safety requirements etc.
27	**Animal_transfer**	transfer_id	Animal_id, from_zoo, transport_mode, transfer_date, transporters	Records details on animals that are transferred from other zoo/national parks.
28	**Insurance**	insurance_id	Animal_id, insurance_company, coverage_items, premium_amount	Holds insurance policy records including insurer details, coverage specifics, premium payments and coverage period.

RELATIONSHIPS

Relationship Name	Entities Involved	Cardinality	Description
Books	Visitor – Ticket	1:M	A visitor can purchase multiple tickets, but each ticket belongs to one visitor.
Includes	Ticket – Tour_Package	M:1	Each ticket corresponds to one tour package, while a tour package can include many tickets.
Leads	Guide – Tour_Package	1:M	Each guide may lead several tour packages; each tour package is handled by one guide.
Uses	Vehicle – Staff	1:M	A vehicle can be assigned to several staff (over time), but a staff uses one vehicle per tour.
Contains	Zone – Habitat	1:M	Each zone contains multiple habitats; each habitat belongs to a single zone.
Houses	enclosure– animal	1:M	Many animals can live in a single enclosure, but each animal is associated with one enclosure.
have	habitat-enclosure	1:M	A habitat can have many enclosures. Each enclosure is consisted in only one habitat.
Assigned_to	Caretaker – enclosure	M:N	Each caretaker takes care of several enclosure; each

			enclosure can have many caretakers assigned.
Maintains	Caretaker – Feeding_Record	1:M	A caretaker creates multiple feeding records; each record is logged by one caretaker.
Maintains	Veterinarian – Health_Record	1:M	Each vet prepares multiple health records; each record is created by one vet.
Uses	Researcher – Permit	1:M	Each researcher can hold multiple permits, but each permit belongs to one researcher.
Handles	Researcher – Conservation_Project	M:N	Researchers can handle multiple conservation projects, and each project can involve many researchers.
Gives	Visitor – Feedback	1:M	A visitor can give multiple feedback entries; each feedback belongs to one visitor.
Belongs_to	Animal-Species	1:M	One animal will belong to only one species, A species may contain many animals
Assigned_to	Animal-Animal_sensor	1:1	One animal will have one sensor, One sensor will be assigned to one animal
belongs_to	insurance-animal	1:1	One animal having one insurance. Each insurance belongs to one animal.

Part_of	Animal-Conservation_project	M:N	An animal can be part of multiple projects, A project can have multiple animals
manages	Supervisor - animal_transfer	1:M	One supervisor can manage many transfer events and each event will be managed by a supervisor
Conducts	Volunteer - Events	M:N	A Volunteers can conduct multiple events, An event may have multiple volunteers
Hosts	Zone - Events	1:M	A zone can host multiple events, An event can be hosted in one zone
Funds	Donation - Conservation Project	1:M	A project can have multiple donations but a donation will fund only one project
Approves	Supervisor - Permit	1:M	A supervisor can approve many permits but a permit is approved by only one supervisor
Manages	Supervisor - Zone	1:M	A supervisor manages multiple zones, A zone will have only one supervisor.
Has	Zone - Tourist Facilities	1:M	A zone can have multiple facilities. A facility belongs to one zone.
Manages	Staff - Tourist Facilities	M:N	A staff manages multiple facilities, a facility may be managed by multiple staff

Manages	Supervisor - Tour Package	1:M	A supervisor can manage multiple packages, A package is managed by one supervisor
Maintains	Staff - Resource Stock	M:N	A staff can maintain records of many stock items. Each stock is maintained by multiple staff.
Takes_part	Visitor - Events	M:N	A visitor can take part in multiple events, An event may have multiple visitors.
has	animal-feeding_record	1:M	a single animal can have many feeding records, while each feeding record belongs to exactly one animal.

DESIGN DECISIONS

The main design goal was to create a **normalized, scalable, and logically organized structure** that reflects real-world activities such as habitat monitoring, animal care, research permissions, visitor management, events, and resource allocation.

1. Domain Structure and Zoning

The system begins with the **Zone** and **Habitat** entities, which reflect how national parks are divided for administrative control and ecological management.
Zones contain habitats having enclosures, which in turn houses animals. This structure ensures:

- ecological hierarchy is maintained
- staff and resource assignment is easier
- reporting can be done zone-wise or habitat-wise

2. Animal & Conservation Focus

Animals are central to park operations, so the database captures detailed information on:

- species classification (via *Species*)
- health (via *Health_Record* & Veterinarians)
- feeding cycles (via *Feeding_Record* & Caretakers)
- conservation efforts (via *Conservation_Project*)

The **Species–Animal** relationship is 1:M to maintain a clear taxonomy.
The **Animal–Sensor** relationship is 1:1 as each animal is assigned a single sensor device for tracking, simplifying monitoring and maintenance.

3. Research and Permit Governance

Researchers and conservation organizations require controlled access.
The system models this through:

- *Permit* entity for authorization
- *Supervisor* approval workflow
- M:N *Researcher–Project* assignment

This ensures both governance and scientific flexibility.

4. Visitor Management and Revenue Tracking

Visitors constitute a major operational and financial component of the park.
The database includes:

- *Visitor*, *Ticket*, and *Tour_Package*
- facilities usage
- feedback and donation tracking

The M:N **Visitor–Event** relationship allows visitors to attend multiple educational or eco-tourism events.

5. Staff Specialization and Responsibilities

Staff are modeled using a parent–child structure:

- Staff → Guide, Caretaker, Veterinarian, Supervisor

This avoids duplication and supports specialized roles.

6. Resource Management

Efficient management of food stocks, vehicles, and facilities is critical.
Thus, entities such as *Vehicle*, *Resource_Stock*, and *Tourist_Facilities* support logistics and ensure that staff can record and track operational data.

7. Events and Community Participation

Events are included to support outreach, tourism, and conservation education.
The system captures:

- volunteer involvement (M:N)
- visitor participation (M:N)
- zones hosting events

Overall the database provides a structured and efficient way to manage wildlife, habitats, staff, visitors, and conservation activities. By capturing essential relationships and enforcing clear constraints, it supports accurate tracking, better decision-making, and smooth park operations.

STAGE:3
Normalization and Schema Design Report

1.ANIMAL

UNF

Attributes
animal_id
name
age
DOB
diet_plan
health_status
gender
enclosure_id
species_id
sensor_id

Primary Key

- animal_id: Unique and NOT NULL

Foreign Keys

- enclosure_id: Must exist in Enclosure(enclosure_id)
- species_id: Must exist in Species(species_id)
- sensor_id: Must exist in Animal_Sensor(sensor_id)

1NF

One animal can have multiple items in its diet_plan. This violates 1NF because a single cell cannot contain a set of values. Therefore we divide the table Animal into Animal and Animal_diet.

ANIMAL

animal_id	name	age	DOB	health_status	gender	enclosure_id	species_id	sensor_id

ANIMAL_DIET

animal_id	diet-item

2. GUIDE

UNF

Attributes
Staff_id
Languages
Guide_rating

Primary Key

- staff_id: Unique and NOT NULL

1 NF

The guide can speak multiple languages which violates 1NF. So divide the Guide table into Guide and Guide_languages.

GUIDE

STAFF_ID	GUIDE_RATING

GUIDE_LANGUAGES

STAFF_ID	LANGUAGES

3. TOUR_PACKAGE

UNF

Attributes
Package_id
Name
Duration
Price
Zone_covered
Guide_id
Sup_id

Primary Key

- package_id: Unique and NOT NULL

Foreign Keys

- guide_id: Must exist in Guide(guide_id / staff_id)
- sup_id: Must exist in Supervisor(sup_id)

1NF

Tour_Package had a multivalued attribute, zone_covered. A package can cover multiple zones. Because a single column cannot contain a set of values (Zone1, Zone2, Zone3), we split tour_package into Tour_Package (entity table)and Package_Zones (junction table).

TOUR_PACKAGE

PACKAGE_ID	NAME	DURATION	PRICE	guide_id	sup_id

PACKAGE_ZONES

PACKAGE_ID	ZONE_ID

4.CONSERVATION_PROJECT

UNF

Attributes
Project_id
Name
Objective
Funding_source

Primary Key

- project_id: Unique and NOT NULL

1NF

A single project can have multiple funding sources (e.g., Government + NGO + Sponsorship).This violates 1NF because a single table cell cannot contain multiple values. So we split Conservation_project into Conservation_project and Project_funding.

CONSERVATION_PROJECT

project_id	name	objective

PROJECT_FUNDING

project_id	funding_source

5. VETENARIAN

UNF

Attributes
Staff_id
License_no
Specialization

Foreign Key (connected to Staff table)

- staff_id: Must exist in Staff(staff_id)

1NF

Storing multiple specializations in one attribute will make it multivalued which violates 1NF. So we split the Vetenarian table into Vetenarian and Vetenarian_specialization.

VETENARIAN

STAFF_ID	LICENSE_NO.

VETENARIAN_SPECIALIZATION

STAFF_ID	SPECIALIZATION

6. HEALTH RECORD

UNF

Attributes
record_id
animal_id
staff_id
treatment
check_up_date

Primary Key

- record_id: Unique and NOT NULL

Foreign Keys

- animal_id: Must exist in Animal(animal_id)
- staff_id: Must exist in Staff(staff_id)

1NF

Treatment attribute could include multiple treatments, which violates 1NF. So we divide it into health_record and health_record_treatment.

HEALTH_RECORD

record_id	animal_id	staff_id	checkup_date

HEALTH_RECORD_TREATMENT

record_id	treatment

7. STAFF

UNF

Attributes
staff_id
name
salary
contact
role
sup_id

Primary Key

- staff_id: Unique and NOT NULL

Foreign Key

- sup_id: Must exist in Supervisor(sup_id)

1NF

Staff could have multiple contact numbers, which violates 1NF. So dividing it into staff and staff_contact.

STAFF

staff_id	name	salary	role	sup_id

STAFF_CONTACT

staff_id	contact

8. ANIMAL_TRANSFER

UNF

Attributes
transfer_id
animal_id
from_zoo
transport_mode
transfer_date
transporters
sup_id

Primary Key

- transfer_id: Unique and NOT NULL

Foreign Keys

- animal_id: Must exist in Animal(animal_id)
- sup_id: Must exist in Supervisor(sup_id)

1NF

Transporters can be multivalued. Therefore divide Animal_transfer table into animal_transfer and transfer_transporter

ANIMAL_TRANSFER

transfer_id	animal_id	from_zoo	transport_mode	transfer_date	sup_id

TRANSFER_TRANSPORTER

transfer_id	transporter_name	transporter_contact

9.INSURANCE

UNF

Attributes
insurance_id
animal_id
insurance_company
company_contact
coverage_items
premium_amount
payment-date
coverage_start_date
coverage_end_date

Primary Key

- insurance_id: Unique and NOT NULL

Foreign Key

- animal_id: Must exist in Animal(animal_id)

INF

Insurance table had multi-valued attributes such as *coverage_items* and *payment_dates* were multivalued. Therefore divide Insurance table into INSURANCE_COVERAGE and INSURANCE_PAYMENT. Each table now contains atomic values, and all repeating groups have been eliminated. Every table has a primary key that uniquely identifies each record.

INSURANCE

insurance_id	animal_id	insurance_company	company_contact	premium_amount	coverage_start_date	coverage_end_date

INSURANCE_COVERAGE

coverage_id	insurance_id	coverage_item

INSURANCE_PAYMENT

payment_id	insurance_id	payment_date

10. RESOURCE_STOCK

UNF

Attributes
stock_id
name
quantity
type
staff_id

Primary Key

- stock_id: Unique and NOT NULL

Foreign Key

- staff_id: Must exist in Staff(staff_id)

1NF

Type can have multiple values. Therefore we divide it into two tables resource_stock and resource_stock_type.

RESOURCE_STOCK

stock_id	name	quantity	staff_id

RESOURCE_STOCK_TYPE

stock_id	type

All the remaining tables—Species, Habitat, Zone, Enclosure, Feeding_Record, Caretaker, Supervisor, Vehicle, Researcher, Donation, Permit, Visitor, Ticket, Tourist_Facility, Event, Volunteer, Feedback, and Animal_Sensor—are already in First Normal Form (1NF) because each table contains only atomic (indivisible) attribute values, has no repeating groups or multivalued attributes, and every field holds a single value for a single record.

2NF

1. TRANSFER_TRANSPORTER

In this table, transfer_id together with transporter_name is the primary key. But transporter_contact depends only on transfer_name. So divide it into transporter and transfer_transporter.

TRANSPORTER

transporter_id	name	contact

TRANSFER_TRANSPORTER

transfer_id	transporter_id

Remaining entities are already in Second Normal Form (2NF) because each table has a single-attribute primary key or a non-composite key, which means there is no possibility of partial dependency. In these tables, every non-key attribute depends entirely on the whole primary key, since the key consists of only one attribute. As a result, no attribute is dependent on just a part of a composite key. Because there are no partial dependencies and every non-key attribute is fully functionally dependent on its primary key, all these entities satisfy the conditions of 2NF.

3NF

1.ANIMAL

Age depends on DOB → derived attribute

Transitive dependency:
animal_id → DOB → age

So we remove age

BEFORE CHANGE

ANIMAL

animal_id	name	DOB	age	health_status	gender

AFTER CHANGE

ANIMAL

animal_id	name	DOB	health_status	gender

2. INSURANCE

In the INSURANCE table, a transitive dependency was identified: company_contact depended on insurance_company instead of depending directly on the primary key (insurance_id). To satisfy 3NF, the attributes insurance_company and company_contact were moved to a new table INSURANCE_COMPANY, and INSURANCE now references it using company_id.

BEFORE CHANGE

INSURANCE

insurance_id	animal_id	insurance_company	company_contact	premium_amount	coverage_start_date	coverage_end_date

AFTER CHANGE

INSURANCE_COMPANY

company_id	insurance_company	company_contact

Remaining entities are already in Third Normal Form (3NF) because they do not contain any transitive dependencies. In each table, every non-key attribute depends directly and only on the primary key, and not on another non-key attribute. There are no attributes that derive their values from other attributes that are not part of the key. Since every non-key attribute is fully functionally dependent on the primary key and there are no indirect (transitive) dependencies, these tables meet all the requirements of 3NF.

Final List Of Tables

ENTITY	ATTRIBUTE	KEYS	FUNCTIONAL DEPENDENCIES
Zone	Zone_id, name, sup_id, habitat_type	PK:zone_id FK:sup_id	zone_id → name, sup_id, habitat_type
Habitat	habitat_id, habitat_name, zone_id	PK:habitat_id FK:zone_id	habitat_id → habitat_name, zone_id
Animal	animal_id, name, gender, health_status, DOB, enclosure_id, species_id, sensor_id	PK:animal_id FK: enclosure_id, species_id , sensor_id	animal_id → name, gender, health_status, DOB, enclosure_id, species_id, sensor_id
Conservation_Project	project_id,name, objective	PK:project_id	project_id → name, objective
Researcher	Res_id, name, organisation, specialisation	PK:Res_id	res_id → name, organisation, specialisation
Permit	permit_id, res_id, purpose, issue_date, valid_till, type, sup_id	PK:permit_id FK:res_id, sup_id	permit_id → res_id, purpose, issue_date, valid_till, type, sup_id

Resource_Stock	Stock_id, name,quantity	PK:Stock_id	stock_id → name, quantity
Staff	staff_id, name, salary, role, sup_id	PK:staff_id FK:sup_id	staff_id → name, salary, role, sup_id
Tourist_Facilities	facility_id, zone_id, facility_type, operating hours	PK:facility_id FK:zone_id	facility_id → zone_id, facility_type, operating_hours
Guide	Staff_id, guide_rating	FK:staff_id	staff_id → guide_rating
Veterinarian	Staff_id, license_no	FK:staff_id	staff_id → license_no
Supervisor	Staff_id, years_of_exp, grade_level, area_of_supervision	FK:staff_id	staff_id → years_of_exp, grade_level, area_of_supervisio n
Caretaker	staff_id, assigned_area, shift_timing	FK:staff_id	staff_id → assigned_area, shift_timing
Enclosure	Enclosure_id,habitat_id , enclosure_name	PK:Enclosure_id FK:habitat_id	enclosure_id → habitat_id, enclosure_name

Ticket	ticket_id, visitor_id, visit_date, payment_mode, type, visitor_count, package_id	PK:ticket_id FK:visitor_id, package_id	ticket_id → visitor_id, visit_date, payment_mode, type, visitor_count, package_id
Tour_Package	package_id, name, duration, guide_id, price, sup_id	PK:package_id FK:guide_id, sup_id	package_id → name, duration, guide_id, price, sup_id
Vehicle	vehicle_id,staff_id,registration_no, usage_purpose, maintenance_date, status	PK:vehicle_id FK:staff_id	vehicle_id → staff_id, registration_no, usage_purpose, maintenance_date, status
Health_Record	record_id, checkup_date animal_id, staff_id	PK:record_id FK:animal_id, staff_id	record_id → checkup_date, animal_id, staff_id
Feeding_Record	record_id, feeding_time, quantity, food_item, animal_id, staff_id	PK:record_id FK:animal_id, staff_id	record_id → feeding_time, quantity, food_item, animal_id, staff_id
Animal_Sensor	Sensor_id, installation_date,battery_level, sensor type,status	PK:Sensor_id	sensor_id → installation_date,

			battery_level, sensor_type, status
Visitor	visitor_id, name, contact_no	PK:visitor_id	visitor_id → name, contact_no
Event	Event_id, Event type, date, duration, zone_id	PK:Event_id FK:zone_id	event_id → event_type, date, duration, zone_id
Volunteer	Volunteer_id, Name, contact_no,assigned_activity,hours_worked	PK:Volunteer_id	volunteer_id → name, contact_no, assigned_activity, hours_worked
Species	Species_id,scientificname,cons_status,common_name,primary_diet_type,avg_lifespan	PK:Species_id	species_id → scientificname, cons_status, common_name, primary_diet_type, avg_lifespan
Feedback	feedback_id, rating, suggestion, date, visitor_id	PK:feedback_id FK:visitor_id	feedback_id → rating, suggestion, date, visitor_id
Donations	donation_id,project_id, Donor name, amount, purpose,date_received	PK:donation_id FK:project_id	donation_id → project_id, donor_name, amount, purpose, date_received

Animal_transfer	Transfer_id, Animal_id, from_zoo, transport_mode, transfer_date, sup_id	PK:Transfer_id FK: animal_id, sup_id	transfer_id → animal_id, from_zoo, transport_mode, transfer_date, sup_id
Insurance	Insurance_id,Animal_id ,premium_amount,com pany_id, coverage_end_date,cov erage_start_date	PK:Insurance_id FK:animal_id, company_id	insurance_id → animal_id, premium_amount, company_id, coverage_end_date , coverage_start_dat e
animal_diet	animal_id, diet_item	PK:animal_id, diet_item FK:animal_id	(animal_id, diet_item) is the composite key. No non-key attributes; each value depends on whole key
Guide_languages	Staff_id, languages	PK:staff_id, languages FK:staff_id	(staff_id, languages) is the composite key
Package_zones	Package_id, zone_id	PK:package_id, zone_id FK:package_id , zone_id	(package_id, zone_id) is the composite key
Project_funding	Project_id, funding_source	PK:project_id, funding_source FK:project_id	project_id → funding_source

Vetenarian_specialization	Staff_id, specialization	PK:staff_id, specialization FK:staff_id	staff_id → specialization
Health_record_treatment	Record_id, treatment	PK:record_id, treatment FK:record_id	record_id → treatment
staff_contact	Staff_id, contact	PK:staff_id, contact FK:staff_id	staff_id → contact
Insurance_coverage	Coverage_id, insurance_id, coverage_item	PK:coverage_id FK:insurance_id	coverage_id → insurance_id, coverage_item
Insurance_payment	Payment_id, insurance_id, payment_date	PK:payment_id FK:insurance_id	payment_id → insurance_id, payment_date
Resource_stock_type	Stock_id, type	PK:stock_id, type FK:stock_id	stock_id → type
Insurance_Company	Company_id, insurance_company, company_contact	PK:company_id	company_id → insurance_company, company_contact
Animal_conservation	Animal_id, project_id	PK:animal_id, project_id FK:animal_id, project_id	(animal_id, project_id)is composite key
Conservetion_researcher	Res_id, project_id	PK:res_id, project_id FK:res_id, project_id	(res_id, project_id) is composite key

Resource_staff	Stock_id, staff_id	PK:stock_id, staff_id FK:stock_id, staff_id	(stock_id, staff_id) is composite key
Tourist_staff	Staff_id, facility_id	PK:staff_id, facility_id FK:staff_id, facility_id	(staff_id, facility_id) is composite key
Enclosure_caretaker	Staff_id, enclosure_id	PK:staff_id, enclosure_id FK:staff_id, enclosure_id	(staff_id, enclosure_id) is composite key
Event_visitor	Visitor_id, event_id	PK:visitor_id, event_id FK:visitor_id, event_id	(visitor_id, event_id)is composite key
Event_volunteer	Event_id, volunteer_id	PK:event_id, volunteer_id FK:event_id, volunteer_id	(event_id, volunteer_id) is composite key

Stage 4:

IMPLEMENTATION AND TESTING REPORT

Table 1: Staff

```
-- 1) Staff (base table for all staff-role subtables)
CREATE TABLE IF NOT EXISTS Staff (
    staff_id INT PRIMARY KEY,
    staff_name VARCHAR(100) NOT NULL,
    salary DECIMAL(12,2) DEFAULT 0.00,
    staff_role VARCHAR(50) NOT NULL,
    sup_id INT DEFAULT NULL, -- optional manager/supervisor (references Staff)
    CONSTRAINT fk_staff_sup
      FOREIGN KEY (sup_id) REFERENCES Staff(staff_id)
      ON DELETE SET NULL ON UPDATE CASCADE
) ;
-- 1) Staff (base table)
INSERT INTO Staff (staff_id, staff_name, salary, staff_role, sup_id) VALUES
(1, 'Alice', 50000, 'Supervisor', NULL),
(2, 'Frank', 28000, 'Supervisor', NULL),
(3, 'Bob', 30000, 'Guide', 1),
(4, 'David', 20000, 'Guide', 1),
(5, 'Charlie', 25000, 'Caretaker', 2),
(6, 'Eve', 22000, 'Caretaker', 2),
(7, 'Grace', 24000, 'Caretaker', 2),
(8, 'Aysel', 20000, 'Guide', 1);
```

Output:

staff_id	staff_name	salary	staff_role	sup_id
1	Alice	50000.00	Supervisor	NULL
2	Frank	28000.00	Supervisor	NULL
3	Bob	30000.00	Guide	1
4	David	20000.00	Guide	1
5	Charlie	25000.00	Caretaker	2
6	Eve	22000.00	Caretaker	2
7	Grace	24000.00	Caretaker	2
8	Aysel	20000.00	Guide	1
NULL	NULL	NULL	NULL	NULL

Table 2: Supervisor

```
-- 2) Supervisor (subtype of Staff)
CREATE TABLE IF NOT EXISTS Supervisor (
    staff_id INT PRIMARY KEY, -- same id as in Staff
    years_of_exp INT DEFAULT 0,
    grade_level VARCHAR(50),
    area_of_supervision VARCHAR(150),
    CONSTRAINT fk_supervisor_staff
      FOREIGN KEY (staff_id) REFERENCES Staff(staff_id)
      ON DELETE CASCADE ON UPDATE CASCADE
) ;
-- 2) Supervisor
INSERT INTO Supervisor (staff_id, years_of_exp, grade_level, area_of_supervision) VALUES
(1, 10, 'A', 'Tourist Facility Zone'),
(2, 8, 'B', 'Safari Zone');
```

Output:

Result Grid | Filter Rows: | Edit:

staff_id	years_of_exp	grade_level	area_of_supervision
1	10	A	Tourist Facility Zone
2	8	B	Safari Zone
NULL	NULL	NULL	NULL

Table 3: Guide

```
-- 3) Guide (subtype of Staff)
CREATE TABLE IF NOT EXISTS Guide (
    staff_id INT PRIMARY KEY,
    guide_rating DECIMAL(3,2) DEFAULT NULL, -- e.g. 4.50
    CONSTRAINT fk_guide_staff
      FOREIGN KEY (staff_id) REFERENCES Staff(staff_id)
      ON DELETE CASCADE ON UPDATE CASCADE
);
-- 3) Guide
INSERT INTO Guide (staff_id, guide_rating) VALUES
(3, 4.5),
(4, 4.0),
(8, 4.2);
```

Output:

staff_id	guide_rating
3	4.50
4	4.00
8	4.20
NULL	NULL

Table 4: Caretaker

```
-- 4) Caretaker (subtype of Staff)
CREATE TABLE IF NOT EXISTS Caretaker (
    staff_id INT PRIMARY KEY,
    assigned_area VARCHAR(150),
    shift_timing VARCHAR(100),
    CONSTRAINT fk_caretaker_staff
      FOREIGN KEY (staff_id) REFERENCES Staff(staff_id)
      ON DELETE CASCADE ON UPDATE CASCADE
);

-- 4) Caretaker
INSERT INTO Caretaker (staff_id, assigned_area, shift_timing) VALUES
(3, 'Safari Zone', 'Morning'),
(5, 'Enclosure Zone', 'Evening'),
(7, 'Safari Zone', 'Afternoon');
```

Output:

staff_id	assigned_area	shift_timing
3	Safari Zone	Morning
5	Enclosure Zone	Evening
7	Safari Zone	Afternoon
NULL	NULL	NULL

Table 5: Zone

```
-- 5) Zone (references Supervisor.staff_id)
CREATE TABLE IF NOT EXISTS Zone (
    zone_id INT PRIMARY KEY,
    zone_name VARCHAR(100) NOT NULL,
    sup_id INT DEFAULT NULL, -- supervisor responsible for zone
    CONSTRAINT fk_zone_supervisor
      FOREIGN KEY (sup_id) REFERENCES Supervisor(staff_id)
      ON DELETE SET NULL ON UPDATE CASCADE
);

-- 5) Zones (administrative)
INSERT INTO Zone (zone_id, zone_name, sup_id) VALUES
(101,'Safari Zone', 1),
(102,'Tourist Facility Zone', 1),
(103,'Enclosure Zone', 1),
(104,'Bird Zone', 2),
(105,'Reptile Zone', 2);
```

Output:

zone_id	zone_name	sup_id
101	Safari Zone	1
102	Tourist Facility Zone	1
103	Enclosure Zone	1
104	Bird Zone	2
105	Reptile Zone	2
NULL	NULL	NULL

Table 6: Tourist Facilities

```sql
-- 6) Tourist Facilities
CREATE TABLE IF NOT EXISTS Tourist_Facilities (
    facility_id INT PRIMARY KEY,
    zone_id INT NOT NULL,
    facility_type VARCHAR(100) NOT NULL,
    operating_hours VARCHAR(100),
    CONSTRAINT fk_facility_zone
      FOREIGN KEY (zone_id) REFERENCES Zone(zone_id)
      ON DELETE CASCADE ON UPDATE CASCADE
      );

-- 6) Tourist Facilities
INSERT INTO Tourist_Facilities (facility_id,zone_id, facility_type, operating_hours) VALUES
(201,102, 'Snack Bar', '09:00-17:00'),
(202,102, 'Gift Shop', '10:00-16:00'),
(203,102, 'Cafe', '08:00-18:00'),
(204,102, 'Souvenir Shop', '09:00-17:00'),
(205,102, 'Photo Booth', '10:00-16:00');
```

Output:

facility_id	zone_id	facility_type	operating_hours
201	102	Snack Bar	09:00-17:00
202	102	Gift Shop	10:00-16:00
203	102	Cafe	08:00-18:00
204	102	Souvenir Shop	09:00-17:00
205	102	Photo Booth	10:00-16:00
NULL	NULL	NULL	NULL

Table 7: Tourist- Facility

```sql
-- 7) Tourist facilities_staff assignment
CREATE TABLE IF NOT EXISTS Tourist_Staff (
    staff_id INT NOT NULL,
    facility_id INT NOT NULL,

    PRIMARY KEY (staff_id, facility_id),

    CONSTRAINT fk_touriststaff_staff
      FOREIGN KEY (staff_id) REFERENCES Staff(staff_id)
      ON DELETE CASCADE ON UPDATE CASCADE,

    CONSTRAINT fk_touriststaff_facility
      FOREIGN KEY (facility_id) REFERENCES Tourist_Facilities(facility_id)
      ON DELETE CASCADE ON UPDATE CASCADE
);
-- 7) Tourist Staff (M:N)
INSERT INTO Tourist_Staff (staff_id, facility_id) VALUES
(2, 201),
(4, 202),
(6, 203),
(2, 204),
(4, 205);
```

Output:

	staff_id	facility_id
▶	2	201
	4	202
	6	203
	2	204
	4	205
*	NULL	NULL

Table 8: Staff- Contact

```
-- 8) staff_contact (multiple contacts per staff allowed)
CREATE TABLE IF NOT EXISTS staff_contact (
    contact_id INT PRIMARY KEY,
    staff_id INT NOT NULL,
    contact VARCHAR(50) NOT NULL,
    contact_type VARCHAR(30) DEFAULT NULL,
    CONSTRAINT fk_staffcontact_staff
      FOREIGN KEY (staff_id) REFERENCES Staff(staff_id)
      ON DELETE CASCADE ON UPDATE CASCADE
);

-- 8) Staff Contacts
INSERT INTO staff_contact (contact_id, staff_id, contact, contact_type) VALUES
(1,1, '9876345112', 'Office'),
(2,1, '6283476587', 'Mobile'),
(3,3, '8871236782', 'Mobile'),
(4,4, '6735476523', 'Mobile'),
(5,5, '9876234665', 'Mobile');
```

Output:

	contact_id	staff_id	contact	contact_type
▶	1	1	9876345112	Office
	2	1	6283476587	Mobile
	3	3	8871236782	Mobile
	4	4	6735476523	Mobile
	5	5	9876234665	Mobile
*	NULL	NULL	NULL	NULL

Table 9: Habitat

```sql
-- 9) Habitat (belongs to a Zone)
CREATE TABLE IF NOT EXISTS Habitat (
    habitat_id INT PRIMARY KEY,
    habitat_name VARCHAR(150) NOT NULL,
    zone_id INT NOT NULL,
    CONSTRAINT fk_habitat_zone
      FOREIGN KEY (zone_id) REFERENCES Zone(zone_id)
      ON DELETE RESTRICT ON UPDATE CASCADE
);

-- 9) Habitat (linked to zones)
INSERT INTO Habitat (habitat_id, habitat_name, zone_id) VALUES
(111, 'Tropical Forest Habitat', 101),    -- Safari Zone
(112, 'Mangrove Wetland Habitat', 103),   -- Enclosure Zone
(113, 'Grassland Habitat', 101),          -- Safari Zone
(114, 'Riverine Habitat', 104),           -- Bird Zone
(115, 'Bird Sanctuary Habitat', 104);     -- Bird Zone
```

Output:

habitat_id	habitat_name	zone_id
111	Tropical Forest Habitat	101
112	Mangrove Wetland Habitat	103
113	Grassland Habitat	101
114	Riverine Habitat	104
115	Bird Sanctuary Habitat	104
NULL	NULL	NULL

Table 10: Enclosure

```sql
-- 10) Enclosure (belongs to a Habitat)
CREATE TABLE IF NOT EXISTS Enclosure (
    enclosure_id INT PRIMARY KEY,
    habitat_id INT NOT NULL,
    enclosure_name VARCHAR(150) NOT NULL,
    CONSTRAINT fk_enclosure_habitat
      FOREIGN KEY (habitat_id) REFERENCES Habitat(habitat_id)
      ON DELETE RESTRICT ON UPDATE CASCADE
);

-- 10) Enclosure
INSERT INTO Enclosure (enclosure_id, habitat_id, enclosure_name) VALUES
(501, 111, 'Lion Enclosure'),
(502, 112, 'Otter Enclosure'),              -- wetland animal
(503, 113, 'Elephant Enclosure'),           -- grassland animal
(504, 114, 'Kingfisher Enclosure'),         -- riverine birds
(505, 115, 'Parrot Aviary');                -- tropical birds
```

Output:

enclosure_id	habitat_id	enclosure_name
501	111	Lion Enclosure
502	112	Otter Enclosure
503	113	Elephant Enclosure
504	114	Kingfisher Enclosure
505	115	Parrot Aviary
NULL	NULL	NULL

Table 11: Species

```
-- 11) Species (independent)
CREATE TABLE IF NOT EXISTS Species (
    species_id INT PRIMARY KEY,
    scientificname VARCHAR(200) NOT NULL,
    cons_status VARCHAR(100),
    common_name VARCHAR(150),
    primary_diet_type VARCHAR(100),
    avg_lifespan INT
);

-- 11) Species
INSERT INTO Species (species_id, scientificname, cons_status, common_name, primary_diet_type, avg_lifespan) VALUES
(1, 'Panthera leo', 'Vulnerable', 'Lion', 'Carnivore', 15),
(2, 'Lutra lutra', 'Near Threatened', 'Otter', 'Carnivore', 12),
(3, 'Elephas maximus', 'Endangered', 'Elephant', 'Herbivore', 60),
(4, 'Alcedo atthis', 'Least Concern', 'Kingfisher', 'Carnivore', 10),
(5, 'Ara macao', 'Least Concern', 'Scarlet Macaw', 'Herbivore', 50);
```

Output:

species_id	scientificname	cons_status	common_name	primary_diet_type	avg_lifespan
1	Panthera leo	Vulnerable	Lion	Carnivore	15
2	Lutra lutra	Near Threatened	Otter	Carnivore	12
3	Elephas maximus	Endangered	Elephant	Herbivore	60
4	Alcedo atthis	Least Concern	Kingfisher	Carnivore	10
5	Ara macao	Least Concern	Scarlet Macaw	Herbivore	50
NULL	NULL	NULL	NULL	NULL	NULL

Table 12: Animal

```
209    -- 12) Animal (references Enclosure and Species)
210
211 •  CREATE TABLE IF NOT EXISTS Animal (
212        animal_id INT PRIMARY KEY,
213        animal_name VARCHAR(120) NOT NULL,
214        gender ENUM('M','F','U') DEFAULT 'U',
215        health_status VARCHAR(100),
216        DOB DATE,
217        enclosure_id INT DEFAULT NULL,
218        species_id INT DEFAULT NULL,
219
220        CONSTRAINT fk_animal_enclosure
221          FOREIGN KEY (enclosure_id) REFERENCES Enclosure(enclosure_id)
222          ON DELETE SET NULL ON UPDATE CASCADE,
223        CONSTRAINT fk_animal_species
224          FOREIGN KEY (species_id) REFERENCES Species(species_id)
225          ON DELETE SET NULL ON UPDATE CASCADE
226    );
227
228 •  INSERT INTO Animal (animal_id, animal_name, gender, health_status, DOB, enclosure_id, species_id) VALUES
229    (1, 'Simba', 'M', 'Healthy', '2010-06-01', 501, 1),    -- Lion
230    (2, 'Ollie', 'F', 'Healthy', '2015-03-12', 502, 2),    -- Otter
231    (3, 'Ella', 'F', 'Healthy', '2008-05-20', 503, 3),     -- Elephant
232    (4, 'Kiko', 'M', 'Healthy', '2018-09-10', 504, 4),     -- Kingfisher
233    (5, 'Coco', 'F', 'Healthy', '2012-11-11', 505, 5);     -- Macaw
```

Output:

animal_id	animal_name	gender	health_status	DOB	enclosure_id	species_id
1	Simba	M	Healthy	2010-06-01	501	1
2	Ollie	F	Healthy	2015-03-12	502	2
3	Ella	F	Healthy	2008-05-20	503	3
4	Kiko	M	Healthy	2018-09-10	504	4
5	Coco	F	Healthy	2012-11-11	505	5
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Table 13: Visitor

```
-- 13) Visitor (independent)
CREATE TABLE IF NOT EXISTS Visitor (
    visitor_id INT PRIMARY KEY,
    visitor_name VARCHAR(120) NOT NULL,
    contact_no VARCHAR(30)
);

-- 13) Visitor
INSERT INTO Visitor (visitor_id,visitor_name, contact_no) VALUES
(121,'John Doe', '9876543210'),
(122,'Jane Smith', '9123456780'),
(123,'Mike Brown', '9988776655'),
(124,'Emma White', '9112233445'),
(125,'Liam Green', '9001122334');
```

Output:

	visitor_id	visitor_name	contact_no
▶	121	John Doe	9876543210
	122	Jane Smith	9123456780
	123	Mike Brown	9988776655
	124	Emma White	9112233445
	125	Liam Green	9001122334
*	NULL	NULL	NULL

Table 14: Tour_package

```
-- 14) Tour_Package (references Guide and Supervisor)
CREATE TABLE IF NOT EXISTS Tour_Package (
    package_id INT PRIMARY KEY,
    package_name VARCHAR(150) NOT NULL,
    duration INT, -- duration in minutes/hours as you prefer
    guide_id INT DEFAULT NULL, -- expects Guide.staff_id
    price DECIMAL(10,2) DEFAULT 0.00,
    sup_id INT DEFAULT NULL, -- supervising staff (Supervisor)
    CONSTRAINT fk_tourpackage_guide
      FOREIGN KEY (guide_id) REFERENCES Guide(staff_id)
      ON DELETE SET NULL ON UPDATE CASCADE,
    CONSTRAINT fk_tourpackage_supervisor
      FOREIGN KEY (sup_id) REFERENCES Supervisor(staff_id)
      ON DELETE SET NULL ON UPDATE CASCADE
);

-- 14) Tour Packages
INSERT INTO Tour_Package (package_id,package_name, duration, guide_id, price, sup_id) VALUES
(111,'Tropical Adventure', 120, 3, 100.00, 1),
(112,'Wetland Trail', 90, 4, 80.00, 1),
(113,'Lion Trek', 150, 8, 120.00, 2),
(114,'Bird Watch', 60, 3, 50.00, 1),
(115,'Reptile Expedition', 90, 8, 90.00, 2);
```

Output:

	package_id	package_name	duration	guide_id	price	sup_id
▶	111	Tropical Adventure	120	3	100.00	1
	112	Wetland Trail	90	4	80.00	1
	113	Lion Trek	150	8	120.00	2
	114	Bird Watch	60	3	50.00	1
	115	Reptile Expedition	90	8	90.00	2
*	NULL	NULL	NULL	NULL	NULL	NULL

Table 15: Package_Zones

```
-- 15) Package_zones (many-to-many: packages <-> zones)
CREATE TABLE IF NOT EXISTS Package_zones (
    id INT AUTO_INCREMENT PRIMARY KEY,
    package_id INT NOT NULL,
    zone_id INT NOT NULL,
    CONSTRAINT uq_package_zone UNIQUE (package_id, zone_id),
    CONSTRAINT fk_packagezones_package
      FOREIGN KEY (package_id) REFERENCES Tour_Package(package_id)
      ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT fk_packagezones_zone
      FOREIGN KEY (zone_id) REFERENCES Zone(zone_id)
      ON DELETE CASCADE ON UPDATE CASCADE
) ;

-- 15) Package Zones (M:N)
INSERT INTO Package_zones (package_id, zone_id) VALUES
(111, 101),
(112, 102),
(113, 101),
(114, 104),
(115, 105);
```

Output:

id	package_id	zone_id
1	111	101
2	112	102
3	113	101
4	114	104
5	115	105
NULL	NULL	NULL

Table 16: Enclosure_Caretaker

```sql
-- 16) Enclosure_caretaker (assign caretakers to enclosures)
CREATE TABLE IF NOT EXISTS Enclosure_caretaker (
    id INT AUTO_INCREMENT PRIMARY KEY,
    staff_id INT NOT NULL,
    enclosure_id INT NOT NULL,
    assigned_from DATE DEFAULT NULL,
    assigned_to DATE DEFAULT NULL,
    CONSTRAINT uq_enclosure_caretaker UNIQUE (staff_id, enclosure_id),
    CONSTRAINT fk_enclosurecaretaker_caretaker
      FOREIGN KEY (staff_id) REFERENCES Caretaker(staff_id)
      ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT fk_enclosurecaretaker_enclosure
      FOREIGN KEY (enclosure_id) REFERENCES Enclosure(enclosure_id)
      ON DELETE CASCADE ON UPDATE CASCADE
);

-- 16) Enclosure Caretakers (M:N)
INSERT INTO Enclosure_caretaker (staff_id, enclosure_id, assigned_from, assigned_to) VALUES
(3, 501, '2025-05-01', '2026-12-31'),
(5, 502, '2025-06-21', '2027-12-31'),
(7, 503, '2025-03-11', '2027-12-31'),
(3, 504, '2025-02-27', '2026-12-31'),
(5, 505, '2025-01-09', '2026-12-31');
```

Output:

	id	staff_id	enclosure_id	assigned_from	assigned_to
▶	1	3	501	2025-05-01	2026-12-31
	2	5	502	2025-06-21	2027-12-31
	3	7	503	2025-03-11	2027-12-31
	4	3	504	2025-02-27	2026-12-31
	5	5	505	2025-01-09	2026-12-31
*	NULL	NULL	NULL	NULL	NULL

Table 17: Ticket

```sql
-- 17) Ticket (references Visitor and Tour_Package)
CREATE TABLE IF NOT EXISTS Ticket (
    ticket_id INT PRIMARY KEY,
    visitor_id INT NOT NULL,
    visit_date DATE NOT NULL,
    payment_mode VARCHAR(50),
    ticket_type VARCHAR(50),
    visitor_count INT DEFAULT 1,
    package_id INT DEFAULT NULL,
    CONSTRAINT fk_ticket_visitor
      FOREIGN KEY (visitor_id) REFERENCES Visitor(visitor_id)
      ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT fk_ticket_package
      FOREIGN KEY (package_id) REFERENCES Tour_Package(package_id)
      ON DELETE SET NULL ON UPDATE CASCADE
);

-- 17) Tickets
INSERT INTO Ticket (ticket_id,visitor_id, visit_date, payment_mode, ticket_type, visitor_count, package_id) VALUES
(610,121, '2025-12-01', 'Credit Card', 'Adult', 2, 111),
(611,122, '2025-12-05', 'Cash', 'Child', 1, 112),
(612,123, '2025-12-10', 'UPI', 'Adult', 3, 113),
(613,124, '2025-12-12', 'Credit Card', 'Adult', 1, 114),
(614,125, '2025-12-15', 'Cash', 'Adult', 2, 115);
```

Output:

ticket_id	visitor_id	visit_date	payment_mode	ticket_type	visitor_count	package_id
610	121	2025-12-01	Credit Card	Adult	2	111
611	122	2025-12-05	Cash	Child	1	112
612	123	2025-12-10	UPI	Adult	3	113
613	124	2025-12-12	Credit Card	Adult	1	114
614	125	2025-12-15	Cash	Adult	2	115
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Queries

1.Which Zone Has the Highest Animal Density?

```sql
-- 1.Which Zone Has the Highest Animal Density?
SELECT Z.zone_name,
       COUNT(A.animal_id)       AS animals_in_zone,
       COUNT(T.ticket_id)       AS visits
FROM Zone Z
LEFT JOIN Habitat H ON Z.zone_id = H.zone_id
LEFT JOIN Enclosure E ON H.habitat_id = E.habitat_id
LEFT JOIN Animal A ON A.enclosure_id = E.enclosure_id
LEFT JOIN Package_zones PZ ON Z.zone_id = PZ.zone_id
LEFT JOIN Tour_Package TP ON PZ.package_id = TP.package_id
LEFT JOIN Ticket T ON TP.package_id = T.package_id
GROUP BY Z.zone_id
ORDER BY animals_in_zone DESC, visits DESC;
```

Output:

zone_name	animals_in_zone	visits
Safari Zone	4	4
Bird Zone	2	2
Enclosure Zone	1	0
Tourist Facility Zone	0	1
Reptile Zone	0	1

2. List all animals with their enclosure, habitat, and zone

```sql
-- 2. List all animals with their enclosure, habitat, and zone
SELECT
    a.animal_id,
    a.animal_name,
    e.enclosure_name,
    h.habitat_name,
    z.zone_name
FROM Animal a
JOIN Enclosure e ON a.enclosure_id = e.enclosure_id
JOIN Habitat h ON e.habitat_id = h.habitat_id
JOIN Zone z ON h.zone_id = z.zone_id;
```

Output:

animal_id	animal_name	enclosure_name	habitat_name	zone_name
1	Simba	Lion Enclosure	Tropical Forest Habitat	Safari Zone
2	Ollie	Otter Enclosure	Mangrove Wetland Habitat	Enclosure Zone
3	Ella	Elephant Enclosure	Grassland Habitat	Safari Zone
4	Kiko	Kingfisher Enclosure	Riverine Habitat	Bird Zone
5	Coco	Parrot Aviary	Bird Sanctuary Habitat	Bird Zone

3. Show supervisors and how many zones they manage

```sql
-- 3. Show supervisors and how many zones they manage
SELECT
    s.staff_name AS supervisor,
    COUNT(z.zone_id) AS zones_managed
FROM Supervisor su
JOIN Staff s ON su.staff_id = s.staff_id
LEFT JOIN Zone z ON su.staff_id = z.sup_id
GROUP BY su.staff_id, s.staff_name;
```

Output:

supervisor	zones_managed
Alice	3
Frank	2

4.Get guides whose rating is above the average guide rating.

```sql
-- 4.Get guides whose rating is above the average guide rating
SELECT
    st.staff_name,
    g.guide_rating
FROM Guide g
JOIN Staff st ON g.staff_id = st.staff_id
WHERE g.guide_rating > (
    SELECT AVG(guide_rating) FROM Guide
);
```

Output:

staff_name	guide_rating
Bob	4.50

5.Revenue generated through tour packages

```sql
-- 5.Revenue generated through tour packages

SELECT tp.package_name AS package_name,
       SUM(t.visitor_count * tp.price) AS total_revenue
FROM Ticket t
JOIN Tour_Package tp ON tp.package_id = t.package_id
GROUP BY tp.package_id;
```

Output:

	package_name	total_revenue
▶	Tropical Adventure	200.00
	Wetland Trail	80.00
	Lion Trek	360.00
	Bird Watch	50.00
	Reptile Expedition	180.00

6. Procedure to register a visitor and book a ticket.

```
CREATE PROCEDURE RegisterVisitorAndTicket(
    IN p_name VARCHAR(100),
    IN p_contact VARCHAR(20),
    IN p_visit_date DATE,
    IN p_payment_mode VARCHAR(50),
    IN p_ticket_type VARCHAR(50),
    IN p_visitor_count INT,
    IN p_package_id INT
)
BEGIN
    DECLARE new_v_id INT;
    DECLARE new_t_id INT;

    SELECT COALESCE(MAX(visitor_id), 0) + 1 INTO new_v_id
    FROM Visitor;
    SELECT COALESCE(MAX(ticket_id), 0) + 1 INTO new_t_id
    FROM Ticket;

    INSERT INTO Visitor (visitor_id, visitor_name, contact_no)
    VALUES (new_v_id, p_name, p_contact);

    INSERT INTO Ticket (ticket_id, visitor_id, visit_date, payment_mode, ticket_type, visitor_count, package_id)
    VALUES (new_t_id, new_v_id, p_visit_date, p_payment_mode, p_ticket_type, p_visitor_count, p_package_id);
END $$
DELIMITER ;
CALL RegisterVisitorAndTicket('Aditi Hydari', '9876543510', '2025-12-09', 'Credit Card', 'Adult', 2, 112);
```

Output:

	ticket_id	visitor_id	visit_date	payment_mode	ticket_type	visitor_count	package_id
▶	610	121	2025-12-01	Credit Card	Adult	2	111
	611	122	2025-12-05	Cash	Child	1	112
	612	123	2025-12-10	UPI	Adult	3	113
	613	124	2025-12-12	Credit Card	Adult	1	114
	614	125	2025-12-15	Cash	Adult	2	115
	615	128	2025-12-01	Credit Card	Adult	2	112
	616	129	2025-12-09	Credit Card	Adult	2	112
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

7. Trigger — Log every new ticket insertion

```
-- 7. Trigger – Log every new ticket insertion
CREATE TABLE Ticket_Log (
  log_id   INT AUTO_INCREMENT PRIMARY KEY,
  ticket_id INT,
  package_id INT,
  visit_date DATETIME,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
DELIMITER //
CREATE TRIGGER after_ticket_insert
AFTER INSERT ON Ticket
FOR EACH ROW
BEGIN
  INSERT INTO Ticket_Log(ticket_id, package_id, visit_date)
  VALUES (NEW.ticket_id, NEW.package_id, NEW.visit_date);
END;
//
DELIMITER ;
```

Output:

log_id	ticket_id	package_id	visit_date	created_at
1	616	112	2025-12-09 00:00:00	2025-11-29 22:45:34
NULL	NULL	NULL	NULL	NULL

Reflection on Learning and Challenges

Throughout the development of this database project, several key challenges shaped our learning experience and contributed meaningfully to our understanding of relational database design. One of the earliest difficulties was defining the appropriate scope of the system. Deciding what operational features of the national park should be represented required thoughtful boundary-setting to ensure the database remained both realistic and manageable. This naturally led into the next challenge: formulating clear, consistent business rules. Translating real-world processes into precise rules highlighted how essential it is to understand workflows before attempting any technical implementation.

Designing the ER diagram was another significant learning milestone. Identifying valid entities and establishing accurate relationships required iterative refinement, especially as we encountered complexities such as multi-level hierarchies, dependency chains, and optional relationships. Determining the correct cardinality for each relationship was particularly demanding. It required careful analysis to ensure the model reflected real-world interactions—whether one-to-one, one-to-many, or many-to-many—without introducing redundancy or violating business constraints.

Normalization presented an additional conceptual challenge. While the theory of 1NF, 2NF, and 3NF is well-established, applying it to our evolving dataset required us to identify hidden dependencies and decide which tables needed decomposition. This process reinforced our understanding of how normalization reduces anomalies while preserving data integrity.

Once the structure stabilized, selecting SQL queries that effectively demonstrated the functionality of the database became an important task. We had to ensure our queries highlighted meaningful administrative operations—such as reporting, validation, aggregation, and multi-table retrieval—while reflecting the practical needs of a national park management system.

Collectively, these challenges not only improved our technical skills but also deepened our appreciation for the systematic thinking required to design a robust and coherent database solution.