LONDON METROPOLITAN UNIVERSITY

islington college
(इस्लिङ्टन कलेज)

CC5051NI - Databases

60% Individual Coursework

2025-26 Spring

Credit: 15 Semester Long Module

Student Name: Eijkeyal Pakhrin

London Met ID:

College ID:

Assignment Due Date: Sunday, January 18, 2026

Assignment Submission Date: Sunday, January 18, 2026

Word Count: 7141

I confirm that I understand my coursework needs to be submitted online via MySecondTeacher under the relevant module page before the deadline in order for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission and a marks of zero will be awarded.

CC5051 Databases Final Coursework Submission

Islington College,Nepal

Document Details

Submission ID
trn:oid:::3618:126385142

Submission Date
Jan 17, 2026, 4:57 PM GMT+5:45

Download Date
Jan 17, 2026, 5:04 PM GMT+5:45

File Name
Final Submission.docx

File Size
1.4 MB

54 Pages

6,184 Words

42,336 Characters

5% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Filtered from the Report

- Bibliography
- Quoted Text
- Cited Text
- Small Matches (less than 50 words)

Exclusions

- 4 Excluded Sources
- 1 Excluded Match

Match Groups

3 Not Cited or Quoted 5%
Matches with neither in-text citation nor quotation marks

0 Missing Quotations 0%
Matches that are still very similar to source material

0 Missing Citation 0%
Matches that have quotation marks, but no in-text citation

0 Cited and Quoted 0%
Matches with in-text citation present, but no quotation marks

Top Sources

0% Internet sources
0% Publications
5% Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

Match Groups

- **3** Not Cited or Quoted 5%
  Matches with neither in-text citation nor quotation marks
- **0** Missing Quotations 0%
  Matches that are still very similar to source material
- **0** Missing Citation 0%
  Matches that have quotation marks, but no in-text citation
- **0** Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

Top Sources

0% Internet sources
0% Publications
5% Submitted works (Student Papers)

Top Sources

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1 Submitted works
Islington College,Nepal on 2026-01-16 **2%**

2 Submitted works
Islington College,Nepal on 2025-12-31 **2%**

3 Submitted works
Islington College,Nepal on 2025-12-31 **<1%**

Table of Contents

Table of Figures

Table of Tables

1. Introduction

1.1. Introduction of the business and its forte

Sajilo Yatra Private Limited is a travel and ticketing company that aims to simplify intercity travel across Nepal by improving transportation connectivity as well as providing a modern, technologydriven booking experience. The company is based in Pokhara and focuses on connecting major cities such as Kathmandu, Pokhara, Chitwan, and Butwal through trusted and organized bus services. Sajilo Yatra is inspired by successful online ticketing platforms seen in other countries, but in recent years the intercity travel sector in Nepal has faced challenges such as a lack of a centralized management system and overpriced fares, routes, and passengers. To fill the gap between passengers and ticketing systems, it is designed specifically for the needs of Nepali passengers and local bus operators and integrates local payment gateways such as Esewa, Khalti, and FonePay. (Narional Express)

Sajilo Yatra was established in 2005 with the vision of connecting the passengers to the destinations while travelling between intercity with a fare price. In the traditional way, people have had to visit multiple counters and bus stops, make phone calls, or rely on local agents to find available buses and seats, especially in peak festivals like Dashain and Tihar. This often creates issues like duplicate bookings of seats, lack of clear information (e.g., time, date, fares), and frustration when buses are cancelled or delayed without prior notice. Here comes Sajilo Yatra. Our mission is to “make intercity travel in Nepal more transparent and dependable through the unified online system.” We value our customer satisfaction, honesty in pricing and policies, and safety using technology to improve everyday life. (RedBus)

Figure 1 sajilo yatra

The forte of Sajilo Yatra and the strength of the database system behind it lie in the centralization of all information and processes related to intercity travel. Despite using separate registers and spreadsheets at different counters. Sajilo Yatra envisions a single integrated platform where routes, trips, buses, staff, passengers, bookings, tickets, and payments are all recorded and managed in one database. This allows passengers to search for trips, check availability, choose booking class type (like regular, premium, and VIP), and receive electronic tickets or confirmation by SMS or mail. At the same time, it must enable the company to assign buses and staff to trips in an organized way, record bus maintenance periods, and track every booking and payment for auditing and reporting.

Sajilo Yatra systems are intended to organize traveling activities, and the database will allow the company to see how many passengers are booked on each trip, which bus is operating on which route, and whether payments have been completed for all tickets issued. This prevents human error, reduces overbookings, and improves communication with passengers through timely notifications about confirmations, delays, or cancellations. To fill this gap, Mr. Kiran Thapa has established SajiloYatra to implement a “modern online booking and management system” for its intercity travel services with a strong and carefully designed database at its core.

1.2. Description of current business

Sajilo Yatra was established in 2010. It operated as a multi-service company handling different bookings. During its initial phase, the company focuses on intercity bus ticketing. It partners with multiple bus operators to provide seat bookings on major routes such as Pokhara–Kathmandu, Kathmandu–Chitwan, and Pokhara–Butwal. However, most of these operators still depend on manual booking and management processes. Staff at various counters receive requests from passengers, check availability by calling operators or consulting paper registers, and then write tickets by hand and record very basic details in simple spreadsheets. In peak seasons like Dashain and Tihar, this can lead to double bookings, uncertainty about exact seat availability and the number of remaining seats left, and confusion about when buses are changed at the last minute.

Apart from managing bus ticket systems, Sajilo Yatra assists passengers with domestic airline ticketing. Employees use the airline system or travel portals to issue tickets, handle cancellations, and process refunds. Customers can visit Sajilo Yatra offices to search for flights, compare fares, and confirm bookings on domestic routes. Moreover, Sajiloyatra also provides tour packages that include lodging and activities for locations like Chitwan and Pokhara.Sajilo Yatra is developing a new intercity travel service that will be completely supported by a modern online booking and management system after realizing that intercity buses have the biggest need for standardization and development. Trips between major cities will be established in this planned service, and each route will have several booked stops with exact arrival and departure

From the point of view of operation or to avoid redundancy, the system must generate tickets with distinct IDs, fares, and issue dates for confirmed bookings; record passenger details; and make the selection of trips and bookings based on class, like premium, VVIP, or general, and the availability of seats, making sure that the total number of seats allotted does not exceed bus capacity. To track ticket fares, it must identify fully paid, partially paid, and unpaid tickets; payments made at physical counters or using online payment gateway systems like eSewa, Khalti, Fonepay, or cards will be linked to relevant tickets. In addition, to avoid a bus being assigned to overlapping trips, the system must handle changes in schedules, like delays or cancellations, and maintain a record of these modifications for future use. Sajilo Yatra made the transition away from manual processes and operates its intercity travel service in a more accurate, dependable, and efficient manner. Mr. Kiran Thapa believes the database design for the company must handle these activities, including buses, routes, trips, passengers, bookings, tickets, payments, and maintenance.

1.3. Business rules

- A passenger may book zero or more trips.
- A trip may have zero or more passengers.
- Each trip must belong to exactly one route.
- A route may have one or more trips.
- Each trip must have one or more buses assigned.
- A bus may be assigned to zero or more trips.
- Each bus assigned to a trip must have one or more staff member assigned to it.
- A staff member may be assigned to zero or more bus-trip combinations.

- Each ticket must be booked by exactly one passenger.
- A passenger may book zero or more tickets.
- A trip may generate zero or more tickets.
- Each ticket must be valid for exactly one trip
- Each ticket may have zero or one payment transaction
- Each payment must be linked to exactly one ticket.
- Each trip follows only one route, there is no intermediate stops are available. (Assumptions)
- Passengers must be uniquely identified using passenger id before booking. (Assumptions)
- Routes can be created even if no trips are scheduled. (Assumptions)
- Availability of seat is calculated by subtracting booked seats from the total capacity of the bus. (Assumptions)
- Staff members are assigned to a specific within a specific trip. (Assumptions)
- Buses marked as 'Under Maintenance' cannot be assigned to trips. (Assumptions)
- At least one bus must be assigned to a trip before tickets can be generated. (Assumptions)
- Payments are single transactions linked to tickets partial payments are not permitted. (Assumptions)
- If payment is not completed within the time limit booking is cancelled and seat is released to another passenger. (Assumptions)
- Fares are based on the travel class like regular, premium and VIP. (Assumptions)
- A trip may exist as an empty schedule before any passenger books tickets. (Assumptions)
- Each ticket reserves one seat on a specific bus for specific trip. (Assumptions)

1.4. Assumptions

- Each trip follows only one route; there is no intermediate stops are available.
- Passengers must be uniquely identified using passenger id before booking.
- Routes can be created even if no trips are scheduled.
- Availability of seat is calculated by subtracting booked seats from the total capacity of the bus.
- Staff members are assigned to a specific within a specific trip.
- Buses marked as 'Under Maintenance' cannot be assigned to trips.
- At least one bus must be assigned to a trip before tickets can be generated.

- Payments are single transactions linked to tickets partial payments are not permitted.
- If payment is not completed within the time limit booking is cancelled and seat is released to another passenger.
- Fares are based on the travel class like regular, premium and VIP.
- A trip may exist as an empty schedule before any passenger books tickets.
- Each ticket reserves one seat on a specific bus for specific trip.

2. Initial ERD

2.1. Identification of Entities and Attributes

Passenger, route, and trip are the three powerful entities that were identified in the first phase. These key entities contributed to developing a "modern online booking and management system" and addressed the case scenario. Attributes with data type and constraint are used to describe each entity.

Entity: Passenger

Table 1: Passenger entity with attributes.

S.NO	Name of Attribute	Data Type	Size	Constraints
1	Passenger_ID	Number	10	Primary Key
2	Full_Name	Character	50	Nut Null
3	Passenger_City	Character	50	Nut Nul
4	Phone_Number	Number	10	Nut Null
5	Email	Character	50	Nut Null
6	Ticket_ID	Number	10	Nut Null
7	Booking Date	Date	-	Nut Null
8	Booking Status	Character	50	-
9	Booking Class	Character	50	-
10	Seat_Number	Character	10	Nut Null
11	Fare_Amount	Number	5,2	Nut Null
12	Payment_ID	Number	10	Nut Null
13	Payment _Method	Character	50	Nut Null
14	Payment_Amount	Number	5,2	Nut Null
15	Payment_Status	Character	50	Nut Null
16	Payment_Datetime	Date	-	Nut Null

Entity: Trip

Table 2: Trip entity with attributes.

S.NO	Name of Attribute	Data Type	Size	Constraints
1	Trip_ID	Number	10	Primary key
2	Departure_Date	Date	-	Nut Null
3	Departure_Time	Date	-	Nut Null
4	Arrival_Time	Date	-	Nut Null
5	Trip_Status	Character	50	Nut Null
6	Bus_ID	Number	10	Nut Null
7	Bus Number	Character	50	Nut Null
8	Bus_Capacity	Number	50	Nut Null
9	Bus_Status	Character	50	-
10	Staff_ID	Number	10	Nut Null
11	Staff_Name	Character	50	Nut Null
12	Staff_Role	Character	50	Nut Null

Entity: Route

Table 3: Route entity with attributes.

S.NO	Name of Attribute	Data Type	Size	Constraints
1	Route_ID	Number	10	Primary Key
2	Route_Name	Character	50	Nut Null
3	Origin_City	Character	50	Nut Null
4	Destiantion_City	Character	50	Nut Null
5	Duration	Number	10	Nut Null

2.2. Initial Entity Relationships diagram

Figure 2 Initial Entity Relationship Diagram

3. Normalization

Normalization is the process of decomposing big tables into many smaller tables to reduce redundancy and remove anomalies from a table. It helps to improve the database's efficiency, consistency and accuracy. It makes databases easier to manage and maintain the data and it ensures that database is adaptable to changing business needs. The objective of normalizations is to eliminate the insertion anomalies, updation anomalies, and deletion anomalies from existing databases. (GeeksForGeek)

3.1. Normalization Process Description

3.1.1. Unnormalized form (UNF)

Un-normalized Form is the simplest and first step of database normalizations which does not meet any of the conditions of database normalization defined by the relational model. (Wikipedia) UNF contains basic form simple utilized obtaining and grouping attributes that requires. All repeating groups of data are indicated using parenthesis (). (DBS211) The unnormalized form of the Sajilo Yatra is given below.

Steps to Apply UNF

- Write down all attributes from the given table or problem.
- Name the entity that contains that contains attributes.
- Identify repeating groups (repeating groups that can have multiple values in a single record).

Trip_Report(Trip_ID,Departure_Date,Departure_Time,Arrival_Time,Trip_Status,Route_ID,Route_Name, Origin_City, Destination_City, Duration, {Bus_ID, Bus_Number, Bus_Capacity, Bus_Status, {Staff_ID, Staff_Name, Staff_Role}{Ticket_ID, Booking_Date, Booking_Status, Booking_Class, Seat_Number,Fare_Amount, Passenger_ID, Full_Name, Passenger_City, Phone_Number,Email, Payment_ID, Payment_Method, Payment_Amount, Payment_Status, Payment_Datetime})

From this unf we see there are multiple repeating groups. Here, I identified repeating groups.

Bus Group → {Bus_ID, Bus_Number, Bus_Capacity, Bus_Status} → One trip can have multiple buses assigned to it.

Staff Group (nested inside bus)→ {Staff_ID, Staff_Name, Staff_Role} → One bus requires multiple staff members to operate. Each bus needs a driver, conductor and helper. When we add multiple staff ffor the same bus, bus information gets repeated.

Ticket Group → {Ticket_ID, Booking_Date, Booking_Status, Booking_Class, Seat_Number, Fare_Amount, Passenger_ID, Full_Name, Passenger_City, Phone_Number, Email, Payment_ID, Payment_Method, Payment_Amount, Payment_Status, Payment_Datetime} → One trip can have multiple tickets booked by different passengers. A bus with multiple seats can have multiple tickets. Each ticket booking repeats the same trip information. Example of Repeating Groups

Problems: Same trip_id and trip_status, Bus_Number repeated multiple times which cause data redundancy causing storage waste and update anomalies.
Solutions: Separated in different tables

3.1.2. First Normal Form(1NF)

First Normal Form (1NF) is the process of removing repeated groups from unnormalized table. In first normal form table should not carry any repeating groups, all fields contain single values, each record is identified uniquely by primary key or candidate key, and tables are properly related using primary key and foreign key relationships. (DBS211).

Steps to apply First Normal Form

The table is said to be in First Normal form (1NF) if and only if.

- All values in each column are atomic (Single Valued).
- All values in a column belong to the same domain.
- There are no repeating groups or multi-valued attributes.

- Each column has a unique name.

Process to convert UNF to 1NF:

Step1: Remove bus repeating group by creating associated entity Trip_Bus table.

Step2: Remove staff repeating group (nested repeating group) by creating Trip_Bus_Staff table.

Step3: Remove ticket repeating group by creating Trip_Ticket table.

Removing Repeating Group in Unormalized Normal Form

Repeating Groups 1: Bus Group

Before Repeating Group exist in UNF

Trip_Report→(Trip_ID,Departure_Date, Departure_Time, Arrival_Time, Trip_Status, Route_ID,Route_Name, Origin_City, Destination_City, Duration, {Bus_ID, Bus_Number, Bus_Capacity, Bus_Status, {Staff_ID, Staff_Name, Staff_Role} }

Problem: one trip contains multiple buses details causing repetition.

Solution: Sperate into another Trip_Bus table.

After Removing Repeating groups in UNF.

Trip_Report → (Trip_ID, Departure_Date, Departure_Time, Arrival_Time, Trip_Status, Route_ID,Route_Name, Origin_City, Destination_City, Duration)

Trip_Bus → (Trip_ID , Bus_ID , Bus_Number, Bus_Capacity, Bus_Status)

Repeating Group 2: Staff Group (Nested Repeating Group)

Before Repeating groups exist in UNF

Trip_Report→(Trip_ID,Departure_Date,Departure_Time,Arrival_Time,Trip_Status,Route_ID, Route_Name, Origin_City, Destination_City, Duration, {Bus_ID, Bus_Number, Bus_Capacity, Bus_Status, {Staff_ID, Staff_Name, Staff_Role })

Problem: One bus on a trip has multiple Staff members assigned.

Solution: Separate in another Trip_Bus_Staff table.

Removing nested repeating groups Fron UNF.

Trip_Report→(Trip_ID,Departure_Date,Departure_Time,Arrival_Time,Trip_Status,Route_ID, Route_Name, Origin_City, Destination_City, Duration)

Trip_Bus → (Trip_ID,Bus_ID,Bus_Number, Bus_Capacity, Bus_Status)

Trip_Bus_Staff→ (Trip_ID , Bus_ID , Staff_ID , Staff_Name, Staff_Role)

Repeating Group 3: Ticket Group

Before removing Repeating groups.

Trip_Report→(Trip_ID,Departure_Date,Departure_Time,Arrival_Time,Trip_Status,Route_ID, Route_Name,Origin_City,Destination_City,Duration,{Ticket_ID,Booking_Date,Booking_Status, Booking_Class,Seat_Number,Fare_Amount,Passenger_ID,Full_Name,Passenger_City, Phone_Number,Email,Payment_ID,Payment_Method,Payment_Amount,Payment_Status, Payment_Datetime}

Problem: One trip has multiple tickets booked by different passengers.

Solution: Separate into Trip_Ticket Table.

Trip_Report→(Trip_ID,Departure_Date,Departure_Time,Arrival_Time,Trip_Status,Route_ID, Route_Name, Origin_City, Destination_City, Duration)

Trip_Bus → (Trip_ID , Bus_ID, Bus_Number, Bus_Capacity, Bus_Status)

Trip_Bus_Staff→ (Trip_ID* , Bus_ID* , Staff_ID , Staff_Name, Staff_Role)

Trip_Ticket → (Ticket_ID, Trip_ID*, Bus_ID*,Booking_Date, Booking_Status, Booking_Class, Seat_Number, Fare_Amount, Passenger_ID, Full_Name, Passenger_City, Phone_Number, Email, Payment_ID, Payment_Method, Payment_Amount, Payment_Status, Payment_Datetime)

First Normal Form Schema

We created one UNF into 4 different tables → Trip (Main table with single valued-attribute) → Trip_Bus (Bus repeating group removed) → Trp_Bus_Staff (Staff repeating group removed)→ Trip_Ticket(Ticket repeating group removed). There is no more repeating groups all values are atomic.

Trip→(Trip_ID,Departure_Date,Departure_Time,Arrival_Time,Trip_Status,Route_ID,Route_Name, Origin_City, Destination_City, Duration)

Trip_Bus → (Trip_ID* , Bus_ID, Bus_Number, Bus_Capacity, Bus_Status)

Trip_Bus_Staff→ (Trip_ID* , Bus_ID *, Staff_ID , Staff_Name, Staff_Role)

Trip_Ticket → (Ticket_ID, Trip_ID*,Bus_ID* , Booking_Date, Booking_Status, Booking_Class, Seat_Number, Fare_Amount, Passenger_ID, Full_Name, Passenger_City, Phone_Number, Email, Payment_ID, Payment_Method, Payment_Amount, Payment_Status, Payment_Datetime)

3.1.3. Second Normal Form (2NF)

Second Normal Form (2NF) is based on the concept of functional dependency. There are two different types of dependencies: full functional dependencies and partial dependencies that we must handle in 2NF. It is a way to organize a database to reduce redundancy and ensure data consistency. Fully functional Dependency means a non-key attribute depends on the entire primary key, not just one part of it. (GeeksForGeek, 2025). Another dependency is Partial

Dependency: In a database, a partial dependency occurs when a non-key column, which is not part of any candidate key, depends on only part of a composite primary key instead of the full key. To minimize these dependencies, every non-key attribute must be fully dependent on the primary key. This ensures that those that are only partially dependent on the primary key are moved to a separate table. This separation enables the database to maintain consistency and operate independently. (GeeksForGeek, 2025).

Steps to apply Second Normal Form

The table is said to be in Second Normal Form (2NF) if and only if.

- It is already in First Normal Form.
- It does not have any partial dependencies (i.e every non-prime attribute is fully functionally dependent on the whole primary key, not just part of a composite key).

What is Partial Dependency?

Partial Dependency occurs when a non-key attribute depends on only part of composite primary key. Partial dependencies only occur when there are multiple composite primary keys exist in tables. Tables with single primary key cannot have partial dependencies.

Example: Primary Key = Trip_ID + Bus_ID

Bus_Number depends on Bus_ID only (not on Trip_ID). This is a Partial Dependencies.

Process to convert 1NF to 2NF.

Step1: Identify tables with composite primary keys.

Step2: Check if non-key attributes depend on whole primary key or part of the primary key.

Step3: If there is any partial dependency found then create separate table.

Step4: Move partially dependent attributes to new table.

Removing Partial Dependency

In the original Trip_Bus table:

Trip_Bus_Table → (Trip_ID, Bus_ID, Bus_Number, Bus_Capacity, Bus_Status)

Composite Primary Key: Trip_ID + Bus_ID

The attributes Bus_Number, Bus_Capacity, Bus_Status only depends on Bus_ID, not both Trip_ID & Bus_ID.

If a bus is used on multiple trips, its details are repeated many times. This causes data redundancy and possible inconsistency.

To remove partial dependency from Trip_Bus

Separate Bus table from trip where Bus_ID act as a primary key.

Bus → (Bus_ID, Bus_Number, Bus_Capacity, Bus_Status)

Trip_Bus → (Trip_ID*, Bus_ID*)

In the original Trip_Bus_Staff Table:

Trip_Bus_Staff → (Trip_ID*, Bus_ID*, Staff_ID, Staff_Name, Staff_Role)

Composite Primary key: Trip_ID + Bus_ID + Staff_ID

The attributes Staff_Name, Staff_Role depend only Staff_ID, not in the whole key. If staff work on multiple buses or trips, their names and roles are repeated in every trip.

To remove partial dependency from Trip_Bus_Staff

Create staff table separately where Staff_ID is the primary key and it records the staff details. Keep Trip_Bus_Staff as bridge entity.

Staff → (Staff_ID, Staff_Name, Staff_Role)

Trip_Bus_Staff → (Trip_ID*, Bus_ID*, Staff_ID*)

Final 2NF schema after removing partial dependencies

To form a Second Normal Form, we identified and removed all partial dependencies.

We separated Bus Details (Bus_Number, Bus_Capacity, Bus_Status) from the Trip_Bus table and placed in a new Bus table.

Also, Staff details (Staff_Name, Staff_Role) were separated from the Trip_Bus_Staff table and placed in a new Staff Table.

Trip→(Trip_ID,Departure_Date,Departure_Time,Arrival_Time,Trip_Status,Route_ID,Route_Name, Origin_City, Destination_City, Duration)

Bus → (Bus_ID, Bus_Number, Bus_Capacity, Bus_Status)

Trip_Bus → (Trip_ID*, Bus_ID*)

Staff → (Staff_ID, Staff_Name, Staff_Role)

Trip_Bus_Staff → (Trip_ID*, Bus_ID*, Staff_ID*)

Trip_Ticket→(Ticket_ID,Trip_ID* , Bus_ID* , Booking_Date, Booking_Status, Booking_Class, Seat_Number, Fare_Amount, Passenger_ID, Full_Name, Passenger_City, Phone_Number, Email, Payment_ID, Payment_Method, Payment_Amount, Payment_Status, Payment_Datetime)

3.1.4. Third Normal Form

Third Normal Form (3NF) is a key concept of database normalization that removes unwanted dependencies. 3NF aims to further reduce redundancy and ensure data integrity by eliminating transitive dependencies. A table is in 3NF if it is already in Second Normal Form (2NF) and has no transitive dependencies, which means that non-key attributes are not dependent on other non-key attributes. To understand Transitive dependency is a situation where a non-key attribute depends on another non-key attribute. To solve 3NF, we eliminate transitive dependencies. reduces redundancy and avoids anomalies during data operations, and it improves data integrity. It ensures that non-key attributes depend only on the primary key. It simplifies maintenance and makes it easier to update and maintain data without affecting other attributes. (Third Normal Form (3NF).

Steps to apply Third Normal Form.

A table is said to be in Third Normal Form (3NF) if and only if.

It is already in Second Normal Form (2NF).

It does not have any transitive dependencies.

i.e, for every functional dependency A → B, either

A is a super key, or B is a prime attribute

What is transitive Dependency

Transitive Dependency occurs when a non-key attribute depends on another non-key attribute, rather than directly on the primary key. This type of dependency usually happens in tables with single primary keys. It may lead to data redundancy and update anomalies.

Example: Primary key = Ticket_ID

Passenger_ID is a non-key attribute.Full_Name depends on Passenger_ID (not directly on Ticket_ID). This is a Transitive Dependency.

Process to Convert Second Normal Form to Third Normal Form

Step1: Identify the tables where non-key attributes depend on other non-key attributes (not directly to the primary key; eg, A→B→C).

Step2: For every transitive dependency found, create a separate table using the associated non-key attribute as its primary key.

Step3: Move the transitive dependent attributes to the new table.

Step4: Update the table to show only references to the new table by key, ensuring that all non-key attributes depend only on the primary key.

Removing Transitive Dependencies from Second Normal Form

In the table Trip_Ticket table:

Trip_Ticket →(Ticket_ID, Trip_ID, Bus_ID, Booking_Date, Booking_Status, Booking_Class, Seat_Number, Fare_Amount, Passenger_ID, Full_Name, Passenger_City, Phone_Number, Email, Payment_ID, Payment_Method, Payment_Amount)

Composite Primary key: Ticket_ID (single column)

The attributes Full_Name, Passenger_City, Phone_Number, Email depend only on Passenger_ID not directly on Ticket_ID. Similarly, Payment_Method, Payment_Amount depend only on Payment_ID not directly on Ticket_ID.

If a passenger buys multiple tickets, their details are repeated for each ticket.

If the payment method or amount changes, it must be updated in multiple places. To remove tansitive Dependencies from Trip_Ticket.

Create a separate passenger table where passenger_ID is the primar key and it records the passenger details.

Create another table for payment and make payment_ID as a primary key and it sotored the payment related details. Keep ticket as the main entity referencing passengers and payment by their IDs.

Passenger → (Passenger_ID, Full_Name, Passenger_City, Phone_Number, Email)

Payment →(Payment_ID, Ticket_ID*, Payment_Method, Payment_Amount)

Ticket → (Ticket_ID, Trip_ID*, Bus_ID*, Booking_Date, Booking_Status, Booking_Class, Seat_Number, Fare_Amount, Passenger_ID*, Payment_ID*)

Now this removes transitive dependencies from 2NF and It Is in Third Normal Form, passenger and payments information is stored only once per entity and ticket simply references these records. All non-key attribute now depends only on the primary key.

Third Normal Form Schema after removing Transitive Dependencies.

Route → (Route_ID, Route_Name, Origin_City, Destination_City, Duration)

Trip → (Trip_ID, Departure_Date, Departure_Time, Arrival_Time, Trip_Status, Route_ID*)

Bus → (Bus_ID, Bus_Number, Bus_Capacity, Bus_Status)

Trip_Bus → (Trip_ID*, Bus_ID*)

Staff → (Staff_ID, Staff_Name, Staff_Role)

Trip_Bus_Staff → (Trip_ID*, Bus_ID*, Staff_ID*)

Ticket→(Ticket_ID,Trip_ID*,Bus_ID*,Booking_Date,Booking_Status,Booking_Class,Seat_Number,Fare_Amount,Passenger_ID*)

Passenger → (Passenger_ID ,Full_Name,Passenger_City,Phone_Number,Email)

Payment → (Payment_ID , Ticket_ID*, Payment_Method,Payment_Amount)

3.2. All Relations Satisfy Third Normal Form (3NF) as Shown by Their Functional Dependencies

1. Route

Attributes: Route_ID, Route_Name, Origin_City, Destination_City, Duration

Functional Dependency exists: Route_ID→Route_Name, Origin_City, Destination_City, Duration

Reason: All non-key attributes depend only on Route_ID, which is the primary key of the table.

2. Trip

Attributes: Trip_ID, Departure_Date, Departure_Time, Arrival_Time, Trip_Status, Route_ID

Functional Dependency exists: Trip_ID→ Departure_Date, Departure_Time, Arrival_Time, Trip_Status, Route_ID*

Reason: All non-key attributes depend only on Trip_ID, which is the primary key for this table.

3. Bus

Attributes: Bus_ID, Bus_Number, Bus_Capacity, Bus_Status

Functional Dependency exists: Bus_ID → Bus_Number, Bus_Capacity, Bus_Status

Reason: All non-key attributes depend only on Bus_ID, which is the primary key of this table.

4. Trip_Bus

Attributes: Trip_ID, Bus_ID

No Functional Dependency exists: Trip_ID, Bus_ID

Reason: This table contains only composite key attributes Trip_ID, Bus_ID as the primary key and there are no non-key attributes.

5. Staff

Attributes: Staff_ID, Staff_Name, Staff_Role

Functional Dependency exists:Staff_ID → Staff_Name, Staff_Role

Reason: All non-key attributes depend only on Staff_ID which is the primary key of this table.

6. Trip_Bus_Staff

Attributes: Trip_ID*, Bus_ID*, Staff_ID*

No Functional Dependency exists:(Trip_ID*, Bus_ID*, Staff_ID*

Reason: This table contains only composite primary key attributes Trip_ID*,Bus_ID*,Staff_ID* as the primary key and there are no primary key attributes.

7. Ticket

Attributes: Ticket_ID, Trip_ID, Bus_ID, Booking_Date, Booking_Status, Booking_Class, Seat_Number, Fare_Amount, Passenger_ID

Functional Dependency exists: Ticket_ID → Trip_ID, Bus_ID, Booking_Date, Booking_Status, Booking_Class, Seat_Number, Fare_Amount, Passenger_ID

Reason: All ther non-key attributes depend only on Ticket_ID which is the primary key of this table.

8. Passenger

Attributes: Passenger_ID, Full_Name, Passenger_City, Phone_Number, Email

Functional Dependency exists:Passenger_ID → Full_Name, Passenger_City, Phone_Number, Email

Reason: All non-key attributes depend only on passenger_ID which is the primary key of this table.

9. Payment

Attributes: Payment_ID, Ticket_ID*, Payment_Method, Payment_Amount

Functional Dependency exists: Payment_ID→Ticket_ID*, Payment_Method, Payment_Amount

Reason: All non-key attributes depend only Payment_ID, which is the primary key of this table.

4. Data Dictionary and Final ERD

4.1. List of entities After Normalization

1.ROUTE

Table 4: Route Data Dictionary.

S.NO	Attribute Name	Data Type	Size	Constraint	Composite Constraint
1	Route_ID	NUMBER	10	PRIMARY KEY	-
2	Route_Name	CHARACTER	50	NOT NULL	-
3	Origin_City	CHARACTER	50	NOT NULL	-
4	Destination_City	CHARACTER	50	NOT NULL	-
5	Duration	NUMBER	10	NOT NULL	-

2.TRIP

Table 5: Trip Data Dictionary.

S.NO	Attribute Name	Data Type	Size	Constraints	Composite Constraints
1	Trip_ID	NUMBER	10	PRIMARY KEY	-
2	Departure_Date	DATE	-	NOT NULL	-
3	Departure_Time	CHARACTER	50	NOT NULL	-
4	Arrival_Time	CHARACTER	50	NOT NULL	-
5	Trip_Status	CHARACTER	50	NOT NULL	-
6	Route_ID	NUMBER	10	FOREIGN KEY	-

3. BUS

Table 6: Bus Data Dictionary.

S.NO	Attribute Name	Data Type	Size	Constraints	Composite Constraints
1	Bus_ID	NUMBER	10	PRIMARY KEY	-
2	Bus_Number	CHARACTER	50	NOT NULL	-
3	Bus_Capacity	NUMBER	3	NOT NULL	-
4	Bus_Status	CHARACTER	15	NOT NULL	-

4.STAFF

Table 7: Staff Data Dictionary.

S.NO	Attribute Name	Data Type	Size	Constraints	Composite Constraints
1	Staff_ID	NUMBER	10	PRIMARY KEY,	-
2	Staff_Name	CHARACTER	50	NOT NULL	-
3	Staff_Role	CHARACTER	50	NOT NULL	-

5.PASSENGER

Table 8: Passenger Data Dictionary.

S.NO	Attribute Name	Data Type	Size	Constraints	Composite Constraints
1	Passenger_ID	NUMBER	10	PRIMARY KEY	-
2	Full_Name	CHARACTER	50	NOT NULL	-
3	Passenger_City	CHARACTER	50	NOT NULL	-
4	Passenger_Number	NUMBER	10	NOT NULL	-
5	Email	CHARACTER	30	NOT NULL	-

7.TRIP_BUS_STAFF

Table 9: Trip_Bus_Staff Data Dictionary.

S.NO	Attribute Name	Data Type	Size	Constraints	Composite Constraints
1	Trip_ID	NUMBER	10	FOREIGN KEY	Primary Key
2	Bus_ID	NUMBER	10	FOREIGN KEY	
3	Staff_ID	NUMBER	10	FOREIGN KEY	

Composite Primary Key: (Trip_ID, Bus_ID, Staff_ID)

8. TICKET

Table 10: Ticket Data Dictionary.

S.NO	Attribute Name	Data Type	Size	Constraints	Composite Constraints
1	Ticket_ID	NUMBER	10	PRIMARY KEY,	-
2	Trip_ID	NUMBER	10	FOREIGN KEY	-
3	Bus_ID	NUMBER	10	FOREIGN KEY	-
4	Booking_Date	DATE	-	NOT NULL	-
5	Booking_Status	CHARACTER	50	NOT NULL	-
6	Booking_Class	CHARACTER	50	NOT NULL	-
7	Seat_Number	NUMBER	3	NOT NULL	-
8	Fare_Amount	NUMBER	5,2	NOT NULL	-
9	Passenger_ID	NUMBER	10	FOREIGN KEY	-

9. PAYMENT

Table 11: Payment Data Dictionary.

S.NO	Attribute Name	Data Type	Size	Constraints	Composite Constraints
1	Payment_ID	NUMBER	10	PRIMARY KEY,	-
2	Ticket_ID	NUMBER	10	FOREIGN KEY	-
3	Payment_Method	CHARACTER	50	NOT NULL	-
4	Payment_Method	NUMBER	10	NOT NULL	-

4.2. Final Entity Relationship Diagram

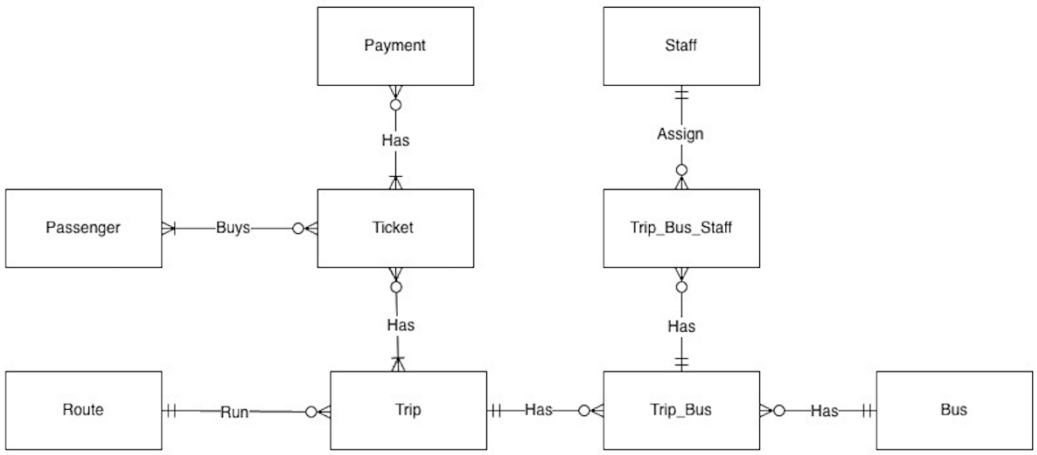

Figure 3 Final Entity Relationship diagram

5. Implementation of Database System

5.1. Creating tables

```
SQL> conn system/test123
Connected.
SQL> CREATE USER EijkeyalPakhrin IDENTIFIED BY 2408849;

User created.

SQL> GRANT CONNECT, RESOURCE TO EijkeyalPakhrin;

Grant succeeded.

SQL>
```

Figure 4 User Created

```
SQL> CONN EijkeyalPakhrin/2408849
Connected.
SQL>
```

Figure 5 User Connected

```
SQL> CREATE TABLE Route (
  2      Route_ID NUMBER(10) PRIMARY KEY,
  3      Route_Name VARCHAR2(30) NOT NULL,
  4      Origin_City VARCHAR2(30) NOT NULL,
  5      Destination_City VARCHAR2(30) NOT NULL,
  6      Duration NUMBER(3) NOT NULL
  7  );

Table created.
```

Figure 6 Creation of Route Table

```
SQL> CREATE TABLE Trip (
  2      Trip_ID NUMBER(10) PRIMARY KEY,
  3      Departure_Date DATE NOT NULL,
  4      Departure_Time VARCHAR2(8) NOT NULL,
  5      Arrival_Time VARCHAR2(8) NOT NULL,
  6      Trip_Status VARCHAR2(15) NOT NULL,
  7      Route_ID NUMBER(10) NOT NULL,
  8      FOREIGN KEY (Route_ID) REFERENCES Route(Route_ID)
  9  );

Table created.
```

Figure 7 Creation of Trip Table

```
SQL> CREATE TABLE Bus (
  2      Bus_ID NUMBER(10) PRIMARY KEY,
  3      Bus_Number VARCHAR2(10) NOT NULL,
  4      Bus_Capacity NUMBER(3) NOT NULL,
  5      Bus_Status VARCHAR2(15) NOT NULL
  6  );

Table created.
```

Figure 8 Creation of Bus Table

```
SQL> CREATE TABLE Staff (
  2      Staff_ID NUMBER(10) PRIMARY KEY,
  3      Staff_Name VARCHAR2(50) NOT NULL,
  4      Staff_Role VARCHAR2(20) NOT NULL
  5  );

Table created.
```

Figure 9 Creation of Staff Table

```
SQL> CREATE TABLE Passenger (
  2       Passenger_ID NUMBER(10) PRIMARY KEY,
  3       Full_Name VARCHAR2(50) NOT NULL,
  4       Passenger_City VARCHAR2(30) NOT NULL,
  5       Phone_Number VARCHAR2(15),
  6       Email VARCHAR2(40)
  7  );

Table created.
```

Figure 10 Creation of Passenger Table

```
SQL> CREATE TABLE Trip_Bus (
  2       Trip_ID NUMBER(10),
  3       Bus_ID NUMBER(10),
  4       PRIMARY KEY (Trip_ID, Bus_ID),
  5       FOREIGN KEY (Trip_ID) REFERENCES Trip(Trip_ID),
  6       FOREIGN KEY (Bus_ID) REFERENCES Bus(Bus_ID)
  7  );

Table created.
```

Figure 11 Creation of Trip_Bus

```
SQL> CREATE TABLE Trip_Bus_Staff (
  2       Trip_ID NUMBER(10),
  3       Bus_ID NUMBER(10),
  4       Staff_ID NUMBER(10),
  5       PRIMARY KEY (Trip_ID, Bus_ID, Staff_ID),
  6       FOREIGN KEY (Trip_ID, Bus_ID) REFERENCES Trip_Bus(Trip_ID, Bus_ID),
  7       FOREIGN KEY (Staff_ID) REFERENCES Staff(Staff_ID)
  8  );

Table created.
```

Figure 12 Creation of Trip_Bus_Staff

```
SQL> CREATE TABLE Ticket (
  2      Ticket_ID NUMBER(10) PRIMARY KEY,
  3      Trip_ID NUMBER(10),
  4      Bus_ID NUMBER(10),
  5      Booking_Date DATE,
  6      Booking_Status VARCHAR2(10),
  7      Booking_Class VARCHAR2(10),
  8      Seat_Number NUMBER(3),
  9      Fare_Amount NUMBER(6),
 10      Passenger_ID NUMBER(10),
 11      FOREIGN KEY (Trip_ID) REFERENCES Trip(Trip_ID),
 12      FOREIGN KEY (Bus_ID) REFERENCES Bus(Bus_ID),
 13      FOREIGN KEY (Passenger_ID) REFERENCES Passenger(Passenger_ID)
 14  );

Table created.
```

Figure 13 Creation of Ticket

```
SQL> CREATE TABLE Payment (
  2      Payment_ID NUMBER(10) PRIMARY KEY,
  3      Ticket_ID NUMBER(10),
  4      Payment_Method VARCHAR2(20),
  5      Payment_Amount NUMBER(10),
  6      FOREIGN KEY (Ticket_ID) REFERENCES Ticket(Ticket_ID)
  7  );

Table created.
```

Figure 14 Creation of Payment

```
SQL> COMMIT;

Commit complete.
```

Figure 15 Commit Successful

5.2. Inserting rows

```
SQL> INSERT INTO Route VALUES (1, 'Kathmandu-Pokhara', 'Kathmandu', 'Pokhara', 6);
1 row created.
SQL> INSERT INTO Route VALUES (2, 'Kathmandu-Biratnagar', 'Kathmandu', 'Biratnagar', 8);
1 row created.
SQL> INSERT INTO Route VALUES (3, 'Kathmandu-Lukla', 'Kathmandu', 'Lukla', 1);
1 row created.
SQL> INSERT INTO Route VALUES (4, 'Pokhara-Chitwan', 'Pokhara', 'Chitwan', 5);
1 row created.
SQL> INSERT INTO Route VALUES (5, 'Biratnagar-Dharan', 'Biratnagar', 'Dharan', 2);
1 row created.
SQL> INSERT INTO Route VALUES (6, 'Kathmandu-Butwal', 'Kathmandu', 'Butwal', 7);
1 row created.
SQL> INSERT INTO Route VALUES (7, 'Pokhara-Lukla', 'Pokhara', 'Lukla', 2);
1 row created.
```

Figure 16 Data Inserted into Route Table

```
SQL> INSERT INTO Trip VALUES (1, DATE '2026-01-20', '06:00:00', '12:00:00', 'Scheduled', 1);
1 row created.
SQL> INSERT INTO Trip VALUES (2, DATE '2026-01-21', '07:00:00', '15:00:00', 'Scheduled', 2);
1 row created.
SQL> INSERT INTO Trip VALUES (3, DATE '2026-01-22', '08:00:00', '09:00:00', 'Scheduled', 3);
1 row created.
SQL> INSERT INTO Trip VALUES (4, DATE '2026-01-23', '06:30:00', '11:30:00', 'Cancelled', 4);
1 row created.
SQL> INSERT INTO Trip VALUES (5, DATE '2026-01-24', '09:00:00', '11:00:00', 'Scheduled', 5);
1 row created.
SQL> INSERT INTO Trip VALUES (6, DATE '2026-01-25', '05:30:00', '12:30:00', 'Scheduled', 6);
1 row created.
SQL> INSERT INTO Trip VALUES (7, DATE '2026-01-26', '10:00:00', '12:00:00', 'Scheduled', 7);
1 row created.
```

Figure 17 Data Inserted into Trip Table

```
SQL> INSERT INTO Bus VALUES (1, 'BA-1001', 40, 'Available');
1 row created.
SQL> INSERT INTO Bus VALUES (2, 'BA-1002', 45, 'Available');
1 row created.
SQL> INSERT INTO Bus VALUES (3, 'BA-1003', 35, 'In Maintenance');
1 row created.
SQL> INSERT INTO Bus VALUES (4, 'BA-1004', 50, 'Available');
1 row created.
SQL> INSERT INTO Bus VALUES (5, 'BA-1005', 30, 'Available');
1 row created.
SQL> INSERT INTO Bus VALUES (6, 'BA-1006', 42, 'Available');
1 row created.
SQL> INSERT INTO Bus VALUES (7, 'BA-1007', 40, 'In Maintenance');
1 row created.
```

Figure 18 Data Inserted into Bus Table

```
SQL> INSERT INTO Staff VALUES (1, 'Ram Sharma', 'Driver');
1 row created.
SQL> INSERT INTO Staff VALUES (2, 'Sita Khadka', 'Conductor');
1 row created.
SQL> INSERT INTO Staff VALUES (3, 'Hari Thapa', 'Driver');
1 row created.
SQL> INSERT INTO Staff VALUES (4, 'Gita Shrestha', 'Conductor');
1 row created.
SQL> INSERT INTO Staff VALUES (5, 'Mohan Rai', 'Driver');
1 row created.
SQL> INSERT INTO Staff VALUES (6, 'Sharmila KC', 'Conductor');
1 row created.
SQL> INSERT INTO Staff VALUES (7, 'Ramesh Adhikari', 'Driver');
1 row created.
```

Figure 19 Data Inserted into Staff Table

```
SQL> INSERT INTO Passenger VALUES (1, 'Anisha Pakhrin', 'Kathmandu', '980000001', 'anisha@mail.com');
1 row created.
SQL> INSERT INTO Passenger VALUES (2, 'Sujan Thapa', 'Pokhara', '980000002', 'sujan@mail.com');
1 row created.
SQL> INSERT INTO Passenger VALUES (3, 'Maya Gurung', 'Biratnagar', '980000003', 'maya@mail.com');
1 row created.
SQL> INSERT INTO Passenger VALUES (4, 'Rohan Shrestha', 'Chitwan', '980000004', 'rohan@mail.com');
1 row created.
SQL> INSERT INTO Passenger VALUES (5, 'Priya KC', 'Lukla', '980000005', 'priya@mail.com');
1 row created.
SQL> INSERT INTO Passenger VALUES (6, 'Anil Rai', 'Butwal', '980000006', 'anil@mail.com');
1 row created.
SQL> INSERT INTO Passenger VALUES (7, 'Suman Sharma', 'Dharan', '980000007', 'suman@mail.com');
1 row created.
```

Figure 20 Data Inserted into Passenger Table

```
SQL> INSERT INTO Trip_Bus VALUES (1, 1);
1 row created.
SQL> INSERT INTO Trip_Bus VALUES (2, 2);
1 row created.
SQL> INSERT INTO Trip_Bus VALUES (3, 3);
1 row created.
SQL> INSERT INTO Trip_Bus VALUES (4, 4);
1 row created.
SQL> INSERT INTO Trip_Bus VALUES (5, 5);
1 row created.
SQL> INSERT INTO Trip_Bus VALUES (6, 6);
1 row created.
SQL> INSERT INTO Trip_Bus VALUES (7, 7);
1 row created.
```

Figure 21 Data Inserted into Trip_Bus Table

```
SQL> INSERT INTO Trip_Bus_Staff VALUES (1, 1, 1);
1 row created.
SQL> INSERT INTO Trip_Bus_Staff VALUES (2, 2, 2);
1 row created.
SQL> INSERT INTO Trip_Bus_Staff VALUES (3, 3, 3);
1 row created.
SQL> INSERT INTO Trip_Bus_Staff VALUES (4, 4, 4);
1 row created.
SQL> INSERT INTO Trip_Bus_Staff VALUES (5, 5, 5);
1 row created.
SQL> INSERT INTO Trip_Bus_Staff VALUES (6, 6, 6);
1 row created.
SQL> INSERT INTO Trip_Bus_Staff VALUES (7, 7, 7);
1 row created.
```

Figure 22 Data Inserted into Trip_Bus_Staff Table

```
SQL> INSERT INTO Ticket VALUES (1, 1, 1, DATE '2026-01-15', 'Confirmed', 'Economy', 1, 1500, 1);
1 row created.
SQL> INSERT INTO Ticket VALUES (2, 2, 2, DATE '2026-01-16', 'Confirmed', 'Economy', 2, 1800, 2);
1 row created.
SQL> INSERT INTO Ticket VALUES (3, 3, 3, DATE '2026-01-17', 'Confirmed', 'Business', 3, 2500, 3);
1 row created.
SQL> INSERT INTO Ticket VALUES (4, 4, 4, DATE '2026-01-18', 'Cancelled', 'Economy', 4, 1200, 4);
1 row created.
SQL> INSERT INTO Ticket VALUES (5, 5, 5, DATE '2026-01-19', 'Confirmed', 'Economy', 5, 1600, 5);
1 row created.
SQL> INSERT INTO Ticket VALUES (6, 6, 6, DATE '2026-01-20', 'Confirmed', 'Business', 6, 3000, 6);
1 row created.
SQL> INSERT INTO Ticket VALUES (7, 7, 7, DATE '2026-01-21', 'Confirmed', 'Economy', 7, 1400, 7);
1 row created.
```

Figure 23 Data Inserted into Ticket Table

```
SQL> ROLLBACK;

Rollback complete.
```

Figure 24 RollBack Successful

```
SQL> INSERT INTO Payment VALUES (1, 1, 'Cash', 1500);
1 row created.
SQL> INSERT INTO Payment VALUES (2, 2, 'Card', 1800);
1 row created.
SQL> INSERT INTO Payment VALUES (3, 3, 'Online', 2500);
1 row created.
SQL> INSERT INTO Payment VALUES (4, 4, 'Cash', 1200);
1 row created.
SQL> INSERT INTO Payment VALUES (5, 5, 'Card', 1600);
1 row created.
SQL> INSERT INTO Payment VALUES (6, 6, 'Online', 3000);
1 row created.
SQL> INSERT INTO Payment VALUES (7, 7, 'Cash', 1400);
1 row created.
```

Figure 25 Data Inserted into Payment Table

```
SQL> COMMIT;

Commit complete.
```

Figure 26 Inserted Data Committed Successfully

```
SQL> SELECT * FROM Route;

  ROUTE_ID ROUTE_NAME                 ORIGIN_CITY                 DESTINATION_CITY              DURATION
---------- ---------------------------- ---------------------------- ---------------------------- ----------
         1 Kathmandu-Pokhara           Kathmandu                    Pokhara                               6
         2 Kathmandu-Biratnagar        Kathmandu                    Biratnagar                            8
         3 Kathmandu-Lukla             Kathmandu                    Lukla                                 1
         4 Pokhara-Chitwan             Pokhara                      Chitwan                               5
         5 Biratnagar-Dharan           Biratnagar                   Dharan                                2
         6 Kathmandu-Butwal            Kathmandu                    Butwal                                7
         7 Pokhara-Lukla               Pokhara                      Lukla                                 2

7 rows selected.
```

Figure 27 Data Displayed from Route Table

```
SQL> SELECT * FROM Trip;

TRIP_ID DEPARTURE DEPARTUR ARRIVAL_ TRIP_STATUS       ROUTE_ID
------- --------- -------- -------- --------------- ----------
      1 20-JAN-26 06:00:00 12:00:00 Scheduled                1
      2 21-JAN-26 07:00:00 15:00:00 Scheduled                2
      3 22-JAN-26 08:00:00 09:00:00 Scheduled                3
      4 23-JAN-26 06:30:00 11:30:00 Cancelled                4
      5 24-JAN-26 09:00:00 11:00:00 Scheduled                5
      6 25-JAN-26 05:30:00 12:30:00 Scheduled                6
      7 26-JAN-26 10:00:00 12:00:00 Scheduled                7

7 rows selected.
```

Figure 28 Data Displayed from Trip Table

```
SQL> SELECT * FROM Bus;

BUS_ID BUS_NUMBER BUS_CAPACITY BUS_STATUS
------ ---------- ------------ ---------------
     1 BA-1001              40 Available
     2 BA-1002              45 Available
     3 BA-1003              35 In Maintenance
     4 BA-1004              50 Available
     5 BA-1005              30 Available
     6 BA-1006              42 Available
     7 BA-1007              40 In Maintenance

7 rows selected.
```

Figure 29 Data Displayed from Bus Table;

```
SQL> SELECT * FROM Staff;

  STAFF_ID STAFF_NAME                                         STAFF_ROLE
---------- -------------------------------------------------- --------------------
         1 Ram Sharma                                         Driver
         2 Sita Khadka                                        Conductor
         3 Hari Thapa                                         Driver
         4 Gita Shrestha                                      Conductor
         5 Mohan Rai                                          Driver
         6 Sharmila KC                                        Conductor
         7 Ramesh Adhikari                                    Driver

7 rows selected.
```

Figure 30 Data Displayed from Staff Table

```
SQL> SELECT * FROM Passenger;

PASSENGER_ID FULL_NAME            PASSENGER_CITY  PHONE_NUMBER    EMAIL
------------ -------------------- --------------- --------------- ------------------------
           1 Anisha Pakhrin       Kathmandu       980000001       anisha@mail.com
           2 Sujan Thapa          Pokhara         980000002       sujan@mail.com
           3 Maya Gurung          Biratnagar      980000003       maya@mail.com
           4 Rohan Shrestha       Chitwan         980000004       rohan@mail.com
           5 Priya KC             Lukla           980000005       priya@mail.com
           6 Anil Rai             Butwal          980000006       anil@mail.com
           7 Suman Sharma         Dharan          980000007       suman@mail.com

7 rows selected.
```

Figure 31 Data Displayed from Passenger Table

```
SQL> SELECT * FROM Trip_Bus;

 TRIP_ID  BUS_ID
-------- -------
       1       1
       2       2
       3       3
       4       4
       5       5
       6       6
       7       7

7 rows selected.
```

Figure 32 Data Displayed from Trip_Bus Table

```
SQL> SELECT * FROM Trip_Bus_Staff;

TRIP_ID  BUS_ID    STAFF_ID
-------- ------- ----------
       1       1          1
       2       2          2
       3       3          3
       4       4          4
       5       5          5
       6       6          6
       7       7          7

7 rows selected.
```

Figure 33 Data Displayed from Trip_Bus_Staff Table

```
SQL> SELECT * FROM Ticket;

TICKET_ID TRIP_ID BUS_ID BOOKING_D BOOKING_ST BOOKING_CL SEAT_NUMBER FARE_AMOUNT PASSENGER_ID
--------- ------- ------ --------- ---------- ---------- ----------- ----------- ------------
        1       1      1 15-JAN-26 Confirmed  Economy              1        1500            1
        2       2      2 16-JAN-26 Confirmed  Economy              2        1800            2
        3       3      3 17-JAN-26 Confirmed  Business             3        2500            3
        4       4      4 18-JAN-26 Cancelled  Economy              4        1200            4
        5       5      5 19-JAN-26 Confirmed  Economy              5        1600            5
        6       6      6 20-JAN-26 Confirmed  Business             6        3000            6
        7       7      7 21-JAN-26 Confirmed  Economy              7        1400            7

7 rows selected.
```

Figure 34 Data Displayed from Ticket Table

```
SQL> SELECT * FROM Payment;

PAYMENT_ID TICKET_ID PAYMENT_METHOD        PAYMENT_AMOUNT
---------- --------- -------------------- --------------
         1         1 Cash                           1500
         2         2 Card                           1800
         3         3 Online                         2500
         4         4 Cash                           1200
         5         5 Card                           1600
         6         6 Online                         3000
         7         7 Cash                           1400

7 rows selected.

SQL> |
```

Figure 35 Data Displayed from Payment

6. Database queries

6.1. Information Query

1. List all routes along with the total number of trips scheduled on each route.

Objective: This query is used to identify all available routes in the system and calculate how many trips are currently scheduled on each route. It returns the Route ID, Route Name, and the total number of trips scheduled in the route.

```
SQL> SELECT r.Route_ID,
  2         r.Route_Name,
  3         COUNT(t.Trip_ID) AS Total_Trips
  4  FROM Route r
  5  LEFT JOIN Trip t ON r.Route_ID = t.Route_ID
  6  GROUP BY r.Route_ID, r.Route_Name
  7  ORDER BY r.Route_ID;

  ROUTE_ID ROUTE_NAME                      TOTAL_TRIPS
---------- ------------------------------ -----------
         1 Kathmandu-Pokhara                        1
         2 Kathmandu-Biratnagar                     1
         3 Kathmandu-Lukla                          1
         4 Pokhara-Chitwan                          1
         5 Biratnagar-Dharan                        1
         6 Kathmandu-Butwal                         1
         7 Pokhara-Lukla                            1

7 rows selected.
```

Figure 36 List of Routes with Total Number of Scheduled Trips

2. List all trips departing from “Pokhara” with their departure time and assigned route name.

Objective: To list all trips that start from Pokhara, including their departure date, departure time, and associated route name. It returns the trip ID, departure date, departure time, and its associated route name for each trip starting in Pokhara.

```
SQL> SELECT t.Trip_ID,
  2         t.Departure_Date,
  3         t.Departure_Time,
  4         r.Route_Name
  5  FROM Trip t
  6  JOIN Route r ON t.Route_ID = r.Route_ID
  7  WHERE r.Origin_City = 'Pokhara'
  8  ORDER BY t.Departure_Date;

TRIP_ID DEPARTURE DEPARTUR ROUTE_NAME
------- --------- -------- ------------------------------
      4 23-JAN-26 06:30:00 Pokhara-Chitwan
      7 26-JAN-26 10:00:00 Pokhara-Lukla
```

Figure 37 Details of Trips Departing from Pokhara

3. **List the names of passengers whose names start with 'A', along with the total number of trips they have booked.**

Objective: To identify passengers whose name begins with the letter ‘A’ and show how many trips each of them has booked. It returns the passenger’s full name along with the total number numbers of tickets booked, even including passengers who may not booked any trips.

```
SQL> SELECT p.Full_Name,
  2         COUNT(t.Ticket_ID) AS Total_Booked_Trips
  3  FROM Passenger p
  4  JOIN Ticket t ON p.Passenger_ID = t.Passenger_ID
  5  WHERE p.Full_Name LIKE 'A%'
  6  GROUP BY p.Full_Name
  7  ORDER BY p.Full_Name;

FULL_NAME            TOTAL_BOOKED_TRIPS
-------------------- ------------------
Anil Rai                              1
Anisha Pakhrin                        1
```

Figure 38 Passengers Starting with 'A' and Their Trip Count

4. **List all passengers who have booked more than 2 trips in December 2025.**

Objective: To find passengers who booked more than two trips during December 2025. This helps identify frequent travelers within a specific time period.

```
SQL> SELECT p.Full_Name,
  2         COUNT(t.Ticket_ID) AS Trips_Booked
  3  FROM Passenger p
  4  JOIN Ticket t ON p.Passenger_ID = t.Passenger_ID
  5  WHERE t.Booking_Date BETWEEN DATE '2025-12-01' AND DATE '2025-12-31'
  6  GROUP BY p.Full_Name
  7  HAVING COUNT(t.Ticket_ID) > 2;

FULL_NAME            TRIPS_BOOKED
-------------------- ------------
Anisha Pakhrin                  3
```

Figure 39 Passengers with More Than Two Trips in December 2025

5. List all tickets issued in January 2025 with passenger names, trip IDs, and payment status ('Paid' or 'Unpaid').

Objective: To list all tickets issued in January 2025 along with passenger names, trips IDs and payment status, indicating whether each ticket has been paid or not.

```
SQL> SELECT t.Ticket_ID,
  2         t.Trip_ID,
  3         p.Full_Name,
  4         CASE
  5             WHEN pay.Payment_ID IS NOT NULL THEN 'Paid'
  6             ELSE 'Unpaid'
  7         END AS Payment_Status
  8  FROM Ticket t
  9  JOIN Passenger p ON t.Passenger_ID = p.Passenger_ID
 10  LEFT JOIN Payment pay ON t.Ticket_ID = pay.Ticket_ID
 11  WHERE t.Booking_Date BETWEEN DATE '2025-01-01' AND DATE '2025-01-31'
 12  ORDER BY t.Ticket_ID;

TICKET_ID TRIP_ID FULL_NAME            PAYMEN
--------- ------- -------------------- ------
       11       4 Sujan Thapa          Unpaid
       12       5 Maya Gurung          Unpaid
```

Figure 40 Tickets Issued in January 2025 with Payment Status

6.2. Transaction Query

1. Find the trip with the latest departure time.

Objective: To identify the trip that depends last based on the departure date and time. This helps determine the most recent scheduled trip. It returns the complete trip record that has the latest combined departure date and departure time in the system.

```
SQL> SELECT *
  2  FROM Trip
  3  WHERE Departure_Date = (SELECT MAX(Departure_Date) FROM Trip);

TRIP_ID DEPARTURE DEPARTUR ARRIVAL_ TRIP_STATUS       ROUTE_ID
------- --------- -------- -------- --------------- ----------
      7 26-JAN-26 10:00:00 12:00:00 Scheduled                7

SQL>
```

Figure 41 Trip with Latest Departure Time

2. List the top three passengers who have booked the most trips.

Objective: To determine the top three passengers who have booked the highest number of trips. This helps identify the most frequent users of the system.

```
SQL> SELECT *
  2  FROM (
  3       SELECT p.Full_Name,
  4              COUNT(t.Ticket_ID) AS Trips_Booked
  5       FROM Passenger p
  6       JOIN Ticket t ON p.Passenger_ID = t.Passenger_ID
  7       GROUP BY p.Full_Name
  8       ORDER BY Trips_Booked DESC
  9  )
 10  WHERE ROWNUM <= 3;

FULL_NAME             TRIPS_BOOKED
--------------------- ------------
Suman Sharma                     1
Maya Gurung                      1
Sujan Thapa                      1
```

Figure 42 Top Three passengers with Most Bookings

3. List all passengers who have made payments greater than the average payment amount, along with the total amount they have paid.

Objective: To list passengers whose total payment amount is greater than the average payment amount. It returns the passenger names and total amount they paid, including only those whose total payments is greater than the average payment amount.

```
SQL> SELECT p.Full_Name,
  2         SUM(pay.Payment_Amount) AS Total_Paid
  3  FROM Passenger p
  4  JOIN Ticket t ON p.Passenger_ID = t.Passenger_ID
  5  JOIN Payment pay ON t.Ticket_ID = pay.Ticket_ID
  6  GROUP BY p.Full_Name
  7  HAVING SUM(pay.Payment_Amount) > (SELECT AVG(Payment_Amount) FROM Payment)
  8  ORDER BY Total_Paid DESC;

FULL_NAME           TOTAL_PAID
------------------- ----------
Anil Rai                  3000
Maya Gurung               2500
```

Figure 43 Passenger with Payments Above the Average Amount

4. Calculate total revenue generated per route along with the number of trips operated on each route.

Objective: To calculate the total revenue generated by each route and the total number of trips operated on that route, and the total revenue generated from ticket fares, ordered from highest to lowest.

```
SQL> SELECT r.Route_Name,
  2         COUNT(DISTINCT t.Trip_ID) AS Total_Trips,
  3         SUM(pay.Payment_Amount) AS Total_Revenue
  4  FROM Route r
  5  JOIN Trip t ON r.Route_ID = t.Route_ID
  6  JOIN Ticket ti ON t.Trip_ID = ti.Trip_ID
  7  JOIN Payment pay ON ti.Ticket_ID = pay.Ticket_ID
  8  GROUP BY r.Route_Name
  9  ORDER BY Total_Revenue DESC;

ROUTE_NAME                     TOTAL_TRIPS TOTAL_REVENUE
------------------------------ ----------- -------------
Kathmandu-Butwal                         1          3000
Kathmandu-Lukla                          1          2500
Kathmandu-Biratnagar                     1          1800
Biratnagar-Dharan                        1          1600
Kathmandu-Pokhara                        1          1500
Pokhara-Lukla                            1          1400
Pokhara-Chitwan                          1          1200

7 rows selected.
```

Figure 44 Total Revenue and Number of Trips per Route

5. List all trips where the number of booked tickets is less than 50% of the bus capacity.

Objective: To identify trips where the number of booked tickets is less than 50% of the total bus capacity.

```
SQL> SELECT t.Trip_ID,
  2         r.Route_Name,
  3         b.Bus_Number,
  4         b.Bus_Capacity,
  5         COUNT(ti.Ticket_ID) AS Tickets_Booked
  6  FROM Trip t
  7  JOIN Route r ON t.Route_ID = r.Route_ID
  8  JOIN Trip_Bus tb ON t.Trip_ID = tb.Trip_ID
  9  JOIN Bus b ON tb.Bus_ID = b.Bus_ID
 10  LEFT JOIN Ticket ti
 11         ON t.Trip_ID = ti.Trip_ID
 12        AND ti.Bus_ID = b.Bus_ID
 13  GROUP BY t.Trip_ID, r.Route_Name, b.Bus_Number, b.Bus_Capacity
 14  HAVING COUNT(ti.Ticket_ID) < (b.Bus_Capacity / 2)
 15  ORDER BY t.Trip_ID;

TRIP_ID ROUTE_NAME                     BUS_NUMBER BUS_CAPACITY TICKETS_BOOKED
------- ------------------------------ ---------- ------------ --------------
      1 Kathmandu-Pokhara              BA-1001              40              1
      2 Kathmandu-Biratnagar           BA-1002              45              1
      3 Kathmandu-Lukla                BA-1003              35              1
      4 Pokhara-Chitwan                BA-1004              50              1
      5 Biratnagar-Dharan              BA-1005              30              1
      6 Kathmandu-Butwal               BA-1006              42              1
      7 Pokhara-Lukla                  BA-1007              40              1

7 rows selected.
```

Figure 45 Trips Less Than 50% Bus Capacity Utilization

7. Critical evaluation

7.1. Critical Evaluation of module, its usage and relation with other subject

This database module has sharpened my coverage of fundamental principles of the database management system. Throughout my learning journey, I have learned valuable skills in database design, normalization techniques, Entity-Relationship Diagram (ERD) development, and SQL implementation. The knowledge that I gained helped my academic as well as professional contexts. Designing a database requires skills that are crucial for roles in software development, system analysis, and data management. In the current scenario, building optimized software is much more important than managing the data correctly through designing the database. The ability that I have to create normalized databases, as demonstrated in my Sajilo Yatra project, ensures data integrity and efficiency in real business scenario systems. These skills are also valuable for developing booking systems, inventory management, and any application requiring structured data storage and retrieval.

The database module is designed by Astha Sharma. I am thankful she actually knows what students need to learn according to the current scenario of the present market. Astha ma'am designed this module knowing student behavior, and she made this as simple and easy to understand and structured as possible. This module covers all the important knowledge for future database designing. She designed this module chunk by chunk. In the very beginning, she introduces database background and history, the need for databases, and the importance of databases and introduces the Oracle database with a basic query, which makes us interact with the database.

After that, in the middle of the module, we deep dive into the core database, like its conceptual level, logical level, and physical level, process by process. We learned about creating business rules and normalization processes. In this module we familiarized ourselves with normalization, why it is needed, and how it helped to reduce redundancy. We learned UNF to 3NF and trap analysis, chasm, and fan. At the end of the module, we learned about designing databases, designing ERDs, and queries to implement in logical databases.

In this module we relate significantly to other subjects within the computing curriculum. It directly connects to software engineering through the significant approach to the system design and development lifecycle. In systems where data is essential for accuracy, this normalizing technique improves system efficiency and removes redundancy. The design of unaltered databases is more crucial in several fields, such as the integration of web development with the database, which serves as the foundation for dynamic websites, data-driven websites, user interactions, and content management. Similar to businesses, data analysis is crucial from the perspective that organizations must have appropriate databases for data creation, transformation, and manipulation in order to grow. As a result, this subject has given me solid foundations between data management and practical database execution in software development, web development, and business analysis, all of which directly support my educational goals and all aspects of computing.

7.2. Critical Assessment of Coursework

This Sajilo Yatra coursework has provided a well-structured opportunity to apply theoretical database design to practical, realistic scenarios. The assignment required the development of a comprehensive database system for Sajilo Yatra, a travel booking company, which effectively simulated real-world business requirements and challenges. The coursework established a clear learning progression. It began with business analysis, moved step-by-step through the stages of normalization, and concluded with SQL implementation. This method effectively connected theoretical learning with practical application.

The main strengths of the coursework included its realistic business scenario, which demanded practical problem-solving and decision-making; its thorough coverage of the entire database development lifecycle; and its balanced approach in requiring both detailed documentation and functional implementation through SQL. I found the normalization process particularly valuable, as progressing from Unnormalized Form to Third Normal Form provided deep insight into eliminating data redundancy and anomalies while ensuring the robust data integrity essential for professional database design. The correct application of normalization up to the Third Normal Form and its referential integrity reflects industry best practices, helping students understand the core concepts of scalable data systems. Using Oracle SQL*Plus immersed

learners in an industry-standard environment, building practical skills relevant to future careers. Completing the assignment within a milestone-driven structure encouraged continuous improvement through feedback, ensuring that good database design involved not only querying but also clear communication and defensible design decisions. The SQL querying section effectively balanced data retrieval and transaction-oriented operations, reinforcing the importance of analytical thinking and the practical application of SQL in business. While effective, the coursework could be strengthened in several areas moving forward. Although the extensive documentation requirements were valuable for ensuring thoroughness, they sometimes led to lengthy explanations rather than concise, industry-style technical documentation. Restricting the coursework to only one database management system, Oracle SQL*Plus, enhanced students' exposure to an enterprise-level relational database environment. However, limiting practice to Oracle may disadvantage students familiar with other platforms such as MySQL, PostgreSQL, or MS SQL, potentially reducing the transferability of their skills unless strongly justified by specific industry needs.

From my personal learning experience, the most challenging aspect was maintaining absolute consistency across the ERD, normalization documentation, and the final SQL implementation. Ensuring all foreign key relationships were correctly implemented and that the normalized tables accurately reflected the initial conceptual design required careful consideration. Successfully navigating this challenge highlighted the crucial importance of careful planning, clear implementation, and systematic verification skills that are directly transferable to professional system development roles.

In summary, this coursework has provided a strong foundation in relational database principles, practical implementation skills, and valuable insights into systematic project development. Overall, it was highly effective in successfully transforming theoretical database concepts into logical database design.

8. Database Schema Export Using Oracle Dump File

8.1. Steps for creating Oracle Dump File

Step1: Opening Command Prompt

The process starts with opening the Windows command prompt, which allows the user to execute database-related commands directly from the operating system.

Step2: Navigating to the Working Directory

Run **cd c:\Users\eijke\OneDrive\Desktop\Second Year\Databases\ Coursework** commands to change the current directory to the folder where the database export file will be stored. This ensures the exported .dmp file is saved in the correct locations.

Step3: Running the Export (exp) command

Run **exp EijkeyalPakhrin/24058849 file = EijkeyalPakhrin_24058849.dmp**. This command initiates the oracle export utility. Define the **oracle username EijkeyalPakhrin**, **password 24058849,** and file, which specifies the name of the dump file that will store the exported database data.

Step4: Successful Completion of Export

Export terminated successfully with warnings. This message confirms that the database export has completed successfully and the .dmp file has been created.

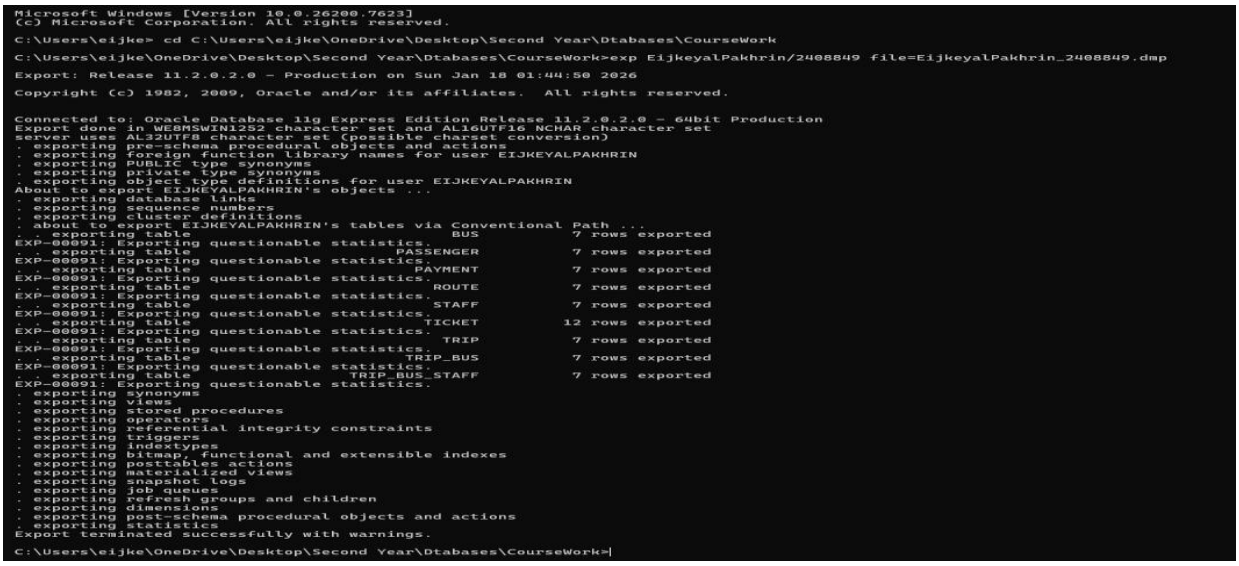

Figure 46 Oracle database export command execution

Step5: Dump file Generated

The dump file **(EijkeyalPakhrin_24058849.dmp)** has been successfully created in the specified directory. This is the screenshot taken to show the generated dump files as proof of successful export for coursework submission.

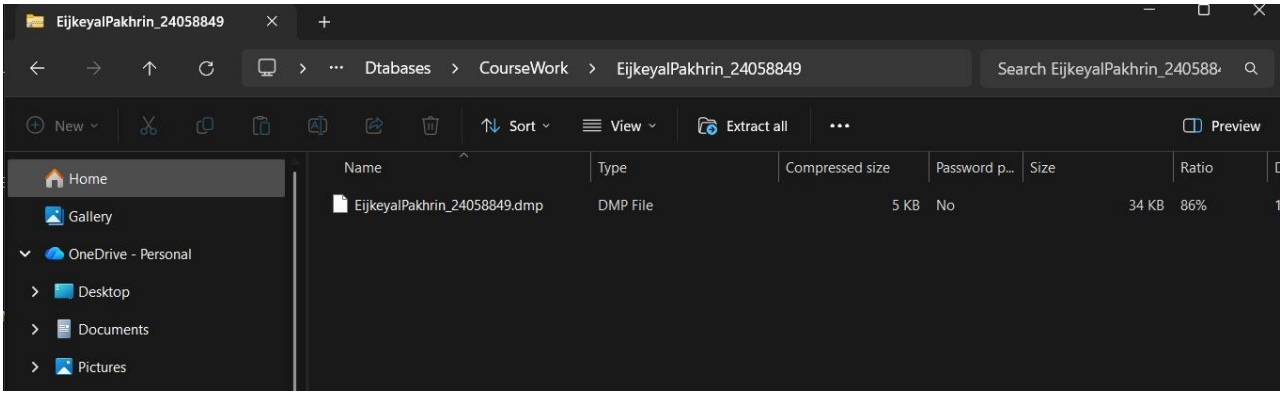

Figure 47 Generated Oracle dump file

8.2. Dropping Database Tables

```
SQL> DROP TABLE Payment;
Table dropped.
SQL> DROP TABLE Ticket;
Table dropped.
SQL> DROP TABLE Trip_Bus_Staff;
Table dropped.
SQL> DROP TABLE Trip_Bus;
Table dropped.
SQL> DROP TABLE Passenger;
Table dropped.
SQL> DROP TABLE Staff;
Table dropped.
SQL> DROP TABLE Bus;
Table dropped.
SQL> DROP TABLE Trip;
Table dropped.
SQL> DROP TABLE Route;
Table dropped.
SQL> |
```

Figure 48 Dropping all Tables

9. References

Dinesh, k. (n.d.). Third Normal Form (3NF). *Medium*. Retrieved December 31, 2025, from https://medium.com/@dinesh.kolli2000/third-normal-form-3nf-aab3e5b4e1e1

GeeksForGeek. (202). *Partial Dependencies.* Retrieved December 31, 2025, from https://www.geeksforgeeks.org/dbms/partial-dependency-in-dbms/

GeeksForGeek. (2025). *Second Normal Form.* GeekForGeek. Retrieved December 31, 2025, from https://www.geeksforgeeks.org/dbms/second-normal-form-2nf/