

|  |  |
| --- | --- |
|  | **AMERICAN INTERNATIONAL UNIVERSITY-BANGLADESH (AIUB)**  **Faculty of Science and Technology Department of Computer Science and Engineering** |

|  |
| --- |
| **FINAL TERM REPORT** |
| **SECTION: D** |

**Advance Database Management System**

























**Project Name: Bus-Ticket Online Booking System**



**Semester: Summer 2022-23**

**Date of Submission: 22/05/20223**

**Submitted to: Rezwan Ahmed Assistant Professor Computer Science Department of CSE**



|  |  |  |  |
| --- | --- | --- | --- |
| **Serial no** | **Student’s Name** | **ID** | **Dept.** |
| 1 | MD MERAZ HOSSAIN | 20-42460-1 | CSE |
| 2 | S. M. ABU HURYRA | 20-42480-1 | CSE |
| 3 | M. KOWSER AHMAD ROBIN | 20-43172-1 | CSE |
| 4 | TAJBIR AHAMMED SHUVO | 20-43796-2 | CSE |

**Table of Contents**

[Sequence 3](#_TOC_250002)

[Table 4](#_TOC_250001)

Searching and Advance Searching [5](#_TOC_250001)

Procedures and Functions 8

[Trigger 10](#_TOC_250001)

[View 13](#_TOC_250001)

[Package 14](#_TOC_250001)



* **Sequences:**

CREATE SEQUENCE "SEQ\_BOOKING\_ID" MINVALUE 1 MAXVALUE 9999999999999999999999999999 INCREMENT BY 1 START WITH 3 NOCACHE NOORDER NOCYCLE NOKEEP NOSCALE GLOBAL ;

CREATE SEQUENCE "SEQ\_CUSTOMER\_ID" MINVALUE 1 MAXVALUE 9999999999999999999999999999 INCREMENT BY 1 START WITH 103 NOCACHE NOORDER NOCYCLE NOKEEP NOSCALE GLOBAL ;

CREATE SEQUENCE "SEQ\_MGR\_ID" MINVALUE 1 MAXVALUE 9999999999999999999999999999 INCREMENT BY 1 START WITH 8 NOCACHE NOORDER NOCYCLE NOKEEP NOSCALE GLOBAL ;

CREATE SEQUENCE "SEQ\_BUS\_ID" MINVALUE 1 MAXVALUE 9999999999999999999999999999 INCREMENT BY 1 START WITH 10 NOCACHE NOORDER NOCYCLE NOKEEP NOSCALE GLOBAL ;

CREATE SEQUENCE "SEQ\_DESTINATION\_ID" MINVALUE 1 MAXVALUE 999999999999999999999 INCREMENT BY 1 START WITH 42 NOCACHE NOORDER

NOCYCLE NOKEEP NOSCALE GLOBAL ;

CREATE SEQUENCE "SEQ\_ROUTE\_ID" MINVALUE 1 MAXVALUE 9999999999999999999999999999 INCREMENT BY 1 START WITH 1 NOCACHE NOORDER NOCYCLE NOKEEP NOSCALE GLOBAL ;

CREATE SEQUENCE "SEQ\_TICKET\_ID" MINVALUE 1 MAXVALUE 9999999999999999999999999999 INCREMENT BY 1 START WITH 61 NOCACHE NOORDER NOCYCLE NOKEEP NOSCALE GLOBAL ;

* **Tables:**

CREATE TABLE "BOOKING"

( "BOOKING\_ID" NUMBER(10,0),

"BUS\_ID" NUMBER(10,0),

"CUSTOMER\_ID" NUMBER(18,0),

"SOURCE\_LOCATION" VARCHAR2(50),

"DESTINATION\_LOCATION" VARCHAR2(50),

"TICKET\_DATE" DATE,

PRIMARY KEY ("BOOKING\_ID")

USING INDEX ENABLE

) ;

CREATE TABLE "BUSES"

( "BUS\_ID" NUMBER(10,0) DEFAULT "WKSP\_AIUB"."SEQ\_BUS\_ID"."NEXTVAL",

"BUS\_NUMBER" VARCHAR2(10),

"BUS\_TYPE" VARCHAR2(20),

"MGR\_ID" NUMBER(10,0),

PRIMARY KEY ("BUS\_ID")

USING INDEX ENABLE

) ;

ALTER TABLE "BUSES" ADD CONSTRAINT "FK\_MGR\_ID" FOREIGN KEY ("MGR\_ID")

REFERENCES "MANAGER" ("MGR\_ID") ENABLE;

CREATE TABLE "CUSTOMER"

( "CUSTOMER\_ID" NUMBER(18,0) DEFAULT "WKSP\_AIUB"."SEQ\_CUSTOMER\_ID"."NEXTVAL",

"CUSTOMER\_NAME" VARCHAR2(50),

"CUSTOMER\_PASS" VARCHAR2(32),

"CUSTOMER\_EMAIL" VARCHAR2(30),

"CUSTOMER\_PHN" VARCHAR2(14),

"MGR\_ID" NUMBER(10,0),

PRIMARY KEY ("CUSTOMER\_ID")

USING INDEX ENABLE

) ;

ALTER TABLE "CUSTOMER" ADD FOREIGN KEY ("MGR\_ID")

REFERENCES "MANAGER" ("MGR\_ID") ENABLE;

CREATE TABLE "DESTINATION"

( "D\_ID" NUMBER DEFAULT "WKSP\_AIUB"."SEQ\_DESTINATION\_ID"."NEXTVAL",

"SOURCE\_LOCATION" VARCHAR2(50),

"DESTINATION\_LOCATION" VARCHAR2(50),

"TICKET\_PRICE" NUMBER,

CONSTRAINT "PK\_DESTINATION" PRIMARY KEY ("D\_ID")

USING INDEX ENABLE

) ;

CREATE TABLE "MANAGER"

( "MGR\_ID" NUMBER(10,0),

"MGR\_PASS" VARCHAR2(32),

"MGR\_NAME" VARCHAR2(20),

"MGR\_EMAIL" VARCHAR2(30),

PRIMARY KEY ("MGR\_ID")

USING INDEX ENABLE

) ;

CREATE TABLE "ROUTE"

( "ROUTE\_ID" NUMBER(10,0) DEFAULT "WKSP\_AIUB"."SEQ\_ROUTE\_ID"."NEXTVAL",

"SOURCE\_LOCATION" VARCHAR2(50),

"DESTINATION\_LOCATION" VARCHAR2(50),

"BUS\_ID" NUMBER(10,0),

PRIMARY KEY ("ROUTE\_ID")

USING INDEX ENABLE

) ;

ALTER TABLE "ROUTE" ADD CONSTRAINT "FK\_ROUTE\_BUS\_ID" FOREIGN KEY ("BUS\_ID")

REFERENCES "BUSES" ("BUS\_ID") ENABLE;

CREATE TABLE "TICKETS"

( "TICKET\_ID" NUMBER(10,0) DEFAULT "WKSP\_AIUB"."SEQ\_TICKET\_ID"."NEXTVAL",

"BUS\_ID" NUMBER(10,0),

"CUSTOMER\_ID" NUMBER(18,0),

"TICKET\_DATE" DATE,

"TICKET\_PRICE" NUMBER(10,2),

PRIMARY KEY ("TICKET\_ID")

USING INDEX ENABLE

) ;

ALTER TABLE "TICKETS" ADD CONSTRAINT "FK\_TICKET\_BUS\_ID" FOREIGN KEY ("BUS\_ID")

REFERENCES "BUSES" ("BUS\_ID") ENABLE;

ALTER TABLE "TICKETS" ADD CONSTRAINT "FK\_TICKET\_CUSTOMER\_ID" FOREIGN KEY ("CUSTOMER\_ID")

REFERENCES "CUSTOMER" ("CUSTOMER\_ID") ENABLE;

CREATE TABLE app\_activity\_log (

log\_id NUMBER GENERATED BY DEFAULT AS IDENTITY,

username VARCHAR2(100),

activity\_date TIMESTAMP,

page\_id NUMBER,

page\_name VARCHAR2(100),

page\_process VARCHAR2(100),

action VARCHAR2(100),

details VARCHAR2(4000)

);

* **Searching and Advance Searching:**

1. SELECT SOURCE\_LOCATION,

ROWID,

DESTINATION\_LOCATION,

TICKET\_PRICE

FROM DESTINATION;

1. SELECT b.\*, d.ticket\_price

FROM booking b

JOIN destination d

ON UPPER(d.source\_location) = UPPER(b.source\_location)

AND UPPER(d.destination\_location) = UPPER(b.destination\_location)

WHERE b.customer\_id = (SELECT customer\_id from customer where UPPER(customer\_name) = UPPER(:APP\_USER));

1. select bk.BOOKING\_ID,

bu.BUS\_NUMBER,

c.CUSTOMER\_NAME,

bk.SOURCE\_LOCATION,

bk.DESTINATION\_LOCATION,

bk.TICKET\_DATE

from BOOKING bk

JOIN BUSES bu ON UPPER(bu.bus\_id) = UPPER(bk.bus\_ID)

JOIN CUSTOMER c ON UPPER (c.customer\_ID) = UPPER(bk.customer\_id);

1. SELECT TO\_CHAR(departure\_time, 'MM') AS month, COUNT(\*) AS booking\_count

FROM (

SELECT ticket\_date as departure\_time

FROM BOOKING

)

GROUP BY TO\_CHAR(departure\_time, 'MM')

ORDER BY month;

1. select b.BUS\_ID,

b.BUS\_NUMBER,

b.BUS\_TYPE,

m.MGR\_NAME

from BUSES b

JOIN MANAGER m ON b.MGR\_ID = m.MGR\_ID;

1. select c.CUSTOMER\_ID,

c.CUSTOMER\_NAME,

c.CUSTOMER\_EMAIL,

c.CUSTOMER\_PHN,

m.MGR\_NAME

from CUSTOMER c

JOIN MANAGER m ON m.MGR\_ID = c.MGR\_ID;

select

MGR\_NAME,

MGR\_EMAIL

from MANAGER;

1. SELECT r.role\_name, COUNT(\*) AS user\_count

FROM (

SELECT 'MANAGER' AS role\_name, mgr\_name AS name FROM MANAGER

UNION ALL

SELECT 'CUSTOMER' AS role\_name, customer\_name AS name FROM CUSTOMER

) r

GROUP BY r.role\_name

ORDER BY COUNT(\*) DESC, r.role\_name;

1. select l.display\_value feedback\_status,

(select count(\*) from apex\_team\_feedback f where f.application\_id = :APP\_ID and f.feedback\_status = l.return\_value) feedback\_count

from apex\_application\_lov\_entries l

where l.application\_id = :APP\_ID

and l.list\_of\_values\_name = 'FEEDBACK\_STATUS'

order by 2 desc, 1;

1. select

trunc(nw.tm - ((level-1)/24),'HH') start\_tm,

trunc(nw.tm - ((level-2)/24),'HH') end\_tm,

trunc(sysdate-((level-1)/24),'HH') log\_start\_tm,

trunc(sysdate-((level-2)/24),'HH') log\_end\_tm

from nw

connect by level <= round( 24 \* ( 1/24/60/60 \* nvl(:P10020\_TIMEFRAME,1) ) )

)

select w.start\_tm log\_time,

( select count(\*)

from apex\_activity\_log l

where l.flow\_id = :app\_id

and l.time\_stamp between w.log\_start\_tm and w.log\_end\_tm ) as value

from window w

order by 1;

1. select nvl( userid\_lc, apex\_lang.message('APEX.FEATURE.TOP\_USERS.USERNAME.NOT\_IDENTIFIED') ) as label,

count(\*) as value

from apex\_activity\_log

where flow\_id = :app\_id

and time\_stamp >= sysdate - ( 1/24/60/60 \* :P10020\_TIMEFRAME )

group by nvl( userid\_lc, apex\_lang.message('APEX.FEATURE.TOP\_USERS.USERNAME.NOT\_IDENTIFIED') )

order by 2 desc;

1. select step\_id,

userid,

time\_stamp err\_time,

sqlerrm,

sqlerrm\_component\_type,

sqlerrm\_component\_name

from apex\_activity\_log

where flow\_id = :app\_id

and sqlerrm is not null;

1. select step\_id page,

( select page\_name

from apex\_application\_pages p

where p.page\_id = l.step\_id

and p.application\_id = :app\_id ) page\_name,

median(elap) median\_elapsed,

count(\*) \* median(elap) weighted\_performance,

sum(decode(sqlerrm,null,0,1)) errors,

count(distinct userid) distinct\_users,

count(distinct session\_id) application\_sessions,

count(\*) page\_views,

max(elap) max\_elapsed,

sum(nvl(num\_rows,0)) total\_rows,

sum(decode(page\_mode,'P',1,0)) partial\_page\_views,

sum(decode(page\_mode,'D',1,0)) full\_page\_views,

min(elap) min\_elapsed,

avg(elap) avg\_elapsed

from apex\_activity\_log l

where flow\_id = :app\_id

and time\_stamp >= sysdate - ( 1/24/60/60 \* :P10023\_TIMEFRAME )

and userid is not null

group by step\_id;

* **Procedures and Functions:**

**User authenticate function:**

create or replace FUNCTION role\_based\_auth(p\_username IN VARCHAR2, p\_password IN VARCHAR2)

RETURN BOOLEAN IS

l\_count INTEGER;

l\_result BOOLEAN := FALSE;

BEGIN

SELECT COUNT(\*)

INTO l\_count

FROM (

SELECT mgr\_name, mgr\_pass FROM MANAGER

UNION ALL

SELECT customer\_name, customer\_pass FROM CUSTOMER

)

WHERE UPPER(mgr\_name) = UPPER(p\_username)

AND UPPER(mgr\_pass) = UPPER(p\_password);

IF l\_count = 1 THEN

l\_result := TRUE;

END IF;

RETURN l\_result;

EXCEPTION

WHEN OTHERS THEN

RETURN FALSE;

END role\_based\_auth;

/

**User role function:**

create or replace FUNCTION get\_user\_role (p\_username IN VARCHAR2) RETURN VARCHAR2 IS

user\_count INTEGER;

BEGIN

-- Check if user is a manager

SELECT COUNT(\*)

INTO user\_count

FROM MANAGER

WHERE UPPER(mgr\_name) = UPPER(p\_username);

IF user\_count = 1 THEN

RETURN 'MANAGER'; -- User is a manager

END IF;

-- If not a manager, check if user is a customer

SELECT COUNT(\*)

INTO user\_count

FROM CUSTOMER

WHERE UPPER(customer\_name) = UPPER(p\_username);

IF user\_count = 1 THEN

RETURN 'CUSTOMER'; -- User is a customer

END IF;

-- If user is neither a manager nor a customer, return 'UNKNOWN'

RETURN 'UNKNOWN';

EXCEPTION

WHEN OTHERS THEN

-- In case of exception, return 'UNKNOWN' and log the error

DBMS\_OUTPUT.PUT\_LINE('An error occurred in get\_user\_role: ' || SQLERRM);

RETURN 'UNKNOWN';

END;

/

**Procedure for inset booking information:**

create or replace PROCEDURE proc\_booking\_insert AS

v\_random\_bus\_id NUMBER;

BEGIN

-- Update BOOKING table with matching bus\_id values

UPDATE BOOKING b

SET bus\_id = NVL(

(

SELECT r.bus\_id

FROM ROUTE r

WHERE r.source\_location = b.source\_location

AND r.destination\_location = b.destination\_location

FETCH FIRST 1 ROWS ONLY

),

FLOOR(DBMS\_RANDOM.VALUE(1, 8)) -- Generate a random number from 1 to 7

)

WHERE b.bus\_id = 0;

-- Commit the changes

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

END;

/

* **Triggers:**

1. CREATE OR REPLACE TRIGGER TRG\_BOOKING\_INSERT

AFTER INSERT ON BOOKING

FOR EACH ROW

BEGIN

-- Update BOOKING table with matching bus\_id values

UPDATE BOOKING b

SET bus\_id = NVL(

(

SELECT r.bus\_id

FROM ROUTE r

WHERE r.source\_location = b.source\_location

AND r.destination\_location = b.destination\_location

FETCH FIRST 1 ROWS ONLY

),

FLOOR(DBMS\_RANDOM.VALUE(1, 8)) -- Generate a random number from 1 to 7

)

WHERE b.bus\_id = 0;

-- Commit the changes

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

-- Handle the exception or log the error here

END;

/

1. CREATE OR REPLACE TRIGGER TRG\_UPDATE\_TICKET\_PRICE

AFTER UPDATE OF TICKET\_PRICE ON DESTINATION

FOR EACH ROW

BEGIN

UPDATE TICKETS

SET TICKET\_PRICE = :NEW.TICKET\_PRICE

WHERE BUS\_ID IN (

SELECT BUS\_ID

FROM ROUTE

WHERE UPPER(SOURCE\_LOCATION) = UPPER(:NEW.SOURCE\_LOCATION)

AND UPPER(DESTINATION\_LOCATION) = UPPER(:NEW.DESTINATION\_LOCATION)

);

DBMS\_OUTPUT.PUT\_LINE('Ticket prices updated successfully.');

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('An error occurred: ' || SQLERRM);

-- You can log the error or perform any necessary error handling here

END;

/

CREATE OR REPLACE TRIGGER TRG\_DELETE\_CUSTOMER

AFTER DELETE ON CUSTOMER

FOR EACH ROW

BEGIN

DELETE FROM TICKETS

WHERE CUSTOMER\_ID = :OLD.CUSTOMER\_ID;

DBMS\_OUTPUT.PUT\_LINE('Related tickets deleted successfully.');

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('An error occurred: ' || SQLERRM);

-- You can log the error or perform any necessary error handling here

END;

/

1. CREATE OR REPLACE TRIGGER trg\_app\_activity\_log

AFTER INSERT OR UPDATE OR DELETE ON apex\_application\_log

FOR EACH ROW

DECLARE

v\_page\_id NUMBER;

v\_page\_name VARCHAR2(100);

v\_page\_process VARCHAR2(100);

v\_action VARCHAR2(100);

v\_details VARCHAR2(4000);

BEGIN

-- Get the page information

SELECT page\_id, page\_name, process INTO v\_page\_id, v\_page\_name, v\_page\_process

FROM apex\_application\_pages

WHERE application\_id = :APP\_ID AND page\_id = :APP\_PAGE\_ID;

-- Set the action based on the type of activity

IF INSERTING THEN

v\_action := 'INSERT';

ELSIF UPDATING THEN

v\_action := 'UPDATE';

ELSIF DELETING THEN

v\_action := 'DELETE';

END IF;

-- Set the details based on the affected rows

IF INSERTING OR UPDATING THEN

v\_details := 'Affected rows: ' || SQL%ROWCOUNT;

ELSIF DELETING THEN

v\_details := 'Deleted rows: ' || SQL%ROWCOUNT;

END IF;

-- Insert the log entry

INSERT INTO app\_activity\_log (username, activity\_date, page\_id, page\_name, page\_process, action, details)

VALUES (:APP\_USER, SYSTIMESTAMP, v\_page\_id, v\_page\_name, v\_page\_process, v\_action, v\_details);

COMMIT;

END;

/

1. CREATE OR REPLACE TRIGGER trg\_generate\_tickets

AFTER INSERT ON BOOKING

FOR EACH ROW

DECLARE

v\_ticket\_id TICKETS.TICKET\_ID%TYPE;

v\_bus\_id TICKETS.BUS\_ID%TYPE;

v\_customer\_id TICKETS.CUSTOMER\_ID%TYPE;

v\_ticket\_date TICKETS.TICKET\_DATE%TYPE;

v\_ticket\_price TICKETS.TICKET\_PRICE%TYPE;

BEGIN

-- Generate values for the TICKETS table based on BOOKING data

v\_ticket\_id := seq\_ticket\_id.NEXTVAL;

v\_bus\_id := :NEW.bus\_id;

v\_customer\_id := :NEW.customer\_id;

v\_ticket\_date := :NEW.ticket\_date;

-- Determine the ticket price based on source and destination locations

-- Lookup the ticket price from the DESTINATION table

SELECT ticket\_price

INTO v\_ticket\_price

FROM DESTINATION

WHERE UPPER(source\_location) = UPPER(:NEW.source\_location)

AND UPPER(destination\_location) = UPPER(:NEW.destination\_location);

-- If no ticket price found, set it to 1000

IF v\_ticket\_price IS NULL THEN

v\_ticket\_price := 1000;

END IF;

-- Insert the generated ticket into the TICKETS table

INSERT INTO TICKETS (TICKET\_ID, BUS\_ID, CUSTOMER\_ID, TICKET\_DATE, TICKET\_PRICE)

VALUES (v\_ticket\_id, v\_bus\_id, v\_customer\_id, v\_ticket\_date, v\_ticket\_price);

DBMS\_OUTPUT.PUT\_LINE('Ticket generated successfully.');

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('An error occurred: ' || SQLERRM);

-- You can log the error or perform any necessary error handling here

END;

/

* **Views:**

1. CREATE VIEW VIEW\_ROUTE\_DETAILS AS

SELECT r.ROUTE\_ID, r.SOURCE\_LOCATION, r.DESTINATION\_LOCATION, b.BUS\_NUMBER

FROM ROUTE r

JOIN BUSES b ON r.BUS\_ID = b.BUS\_ID;

1. CREATE VIEW VIEW\_MANAGER\_CUSTOMERS AS

SELECT m.MGR\_NAME, m.MGR\_EMAIL, c.CUSTOMER\_NAME, c.CUSTOMER\_EMAIL

FROM MANAGER m

JOIN CUSTOMER c ON m.MGR\_ID = c.MGR\_ID;

1. CREATE VIEW VIEW\_DESTINATION\_PRICES AS

SELECT d.SOURCE\_LOCATION, d.DESTINATION\_LOCATION, d.TICKET\_PRICE

FROM DESTINATION d;

1. CREATE VIEW VIEW\_BOOKING\_SUMMARY AS

SELECT b.BOOKING\_ID, b.TICKET\_DATE, c.CUSTOMER\_NAME, d.SOURCE\_LOCATION, d.DESTINATION\_LOCATION

FROM BOOKING b

JOIN CUSTOMER c ON b.CUSTOMER\_ID = c.CUSTOMER\_ID

JOIN DESTINATION d ON (d.SOURCE\_LOCATION = b.SOURCE\_LOCATION AND d.DESTINATION\_LOCATION = b.DESTINATION\_LOCATION);

1. CREATE VIEW VIEW\_CUSTOMER\_DETAILS AS

SELECT c.CUSTOMER\_ID, c.CUSTOMER\_NAME, c.CUSTOMER\_EMAIL, b.BUS\_NUMBER

FROM CUSTOMER c

LEFT JOIN BUSES b ON c.MGR\_ID = b.MGR\_ID;

* **Packages:**

CREATE OR REPLACE PACKAGE PKG\_BOOKING\_PROCESS IS

PROCEDURE insert\_booking(

p\_bus\_id IN BOOKING.BUS\_ID%TYPE,

p\_customer\_id IN BOOKING.CUSTOMER\_ID%TYPE,

p\_source\_location IN BOOKING.SOURCE\_LOCATION%TYPE,

p\_destination\_location IN BOOKING.DESTINATION\_LOCATION%TYPE,

p\_ticket\_date IN BOOKING.TICKET\_DATE%TYPE

);

PROCEDURE generate\_tickets(

p\_booking\_id IN BOOKING.BOOKING\_ID%TYPE

);

END PKG\_BOOKING\_PROCESS;

/

CREATE OR REPLACE PACKAGE BODY PKG\_BOOKING\_PROCESS IS

PROCEDURE insert\_booking(

p\_bus\_id IN BOOKING.BUS\_ID%TYPE,

p\_customer\_id IN BOOKING.CUSTOMER\_ID%TYPE,

p\_source\_location IN BOOKING.SOURCE\_LOCATION%TYPE,

p\_destination\_location IN BOOKING.DESTINATION\_LOCATION%TYPE,

p\_ticket\_date IN BOOKING.TICKET\_DATE%TYPE

) IS

v\_booking\_id BOOKING.BOOKING\_ID%TYPE;

BEGIN

-- Insert a new booking record

INSERT INTO BOOKING (

BOOKING\_ID,

BUS\_ID,

CUSTOMER\_ID,

SOURCE\_LOCATION,

DESTINATION\_LOCATION,

TICKET\_DATE

) VALUES (

SEQ\_BOOKING\_ID.NEXTVAL,

p\_bus\_id,

p\_customer\_id,

p\_source\_location,

p\_destination\_location,

p\_ticket\_date

)

RETURNING BOOKING\_ID INTO v\_booking\_id;

DBMS\_OUTPUT.PUT\_LINE('Booking inserted successfully. Booking ID: ' || v\_booking\_id);

-- Generate tickets for the booking

generate\_tickets(v\_booking\_id);

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('An error occurred in insert\_booking: ' || SQLERRM);

-- You can log the error or perform any necessary error handling here

END insert\_booking;

PROCEDURE generate\_tickets(

p\_booking\_id IN BOOKING.BOOKING\_ID%TYPE

) IS

v\_bus\_id TICKETS.BUS\_ID%TYPE;

v\_customer\_id TICKETS.CUSTOMER\_ID%TYPE;

v\_ticket\_date TICKETS.TICKET\_DATE%TYPE;

v\_ticket\_price TICKETS.TICKET\_PRICE%TYPE;

BEGIN

-- Retrieve booking details

SELECT BUS\_ID, CUSTOMER\_ID, TICKET\_DATE

INTO v\_bus\_id, v\_customer\_id, v\_ticket\_date

FROM BOOKING

WHERE BOOKING\_ID = p\_booking\_id;

-- Determine the ticket price based on source and destination locations

-- Lookup the ticket price from the DESTINATION table

SELECT TICKET\_PRICE

INTO v\_ticket\_price

FROM DESTINATION

WHERE UPPER(SOURCE\_LOCATION) = UPPER((SELECT SOURCE\_LOCATION FROM BOOKING WHERE BOOKING\_ID = p\_booking\_id))

AND UPPER(DESTINATION\_LOCATION) = UPPER((SELECT DESTINATION\_LOCATION FROM BOOKING WHERE BOOKING\_ID = p\_booking\_id));

-- If no ticket price found, set it to 1000

IF v\_ticket\_price IS NULL THEN

v\_ticket\_price := 1000;

END IF;

-- Insert the generated ticket into the TICKETS table

INSERT INTO TICKETS (TICKET\_ID, BUS\_ID, CUSTOMER\_ID, TICKET\_DATE, TICKET\_PRICE)

VALUES (SEQ\_TICKET\_ID.NEXTVAL, v\_bus\_id, v\_customer\_id, v\_ticket\_date, v\_ticket\_price);

DBMS\_OUTPUT.PUT\_LINE('Ticket generated successfully.');

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('No ticket price found for the booking.');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('An error occurred in generate\_tickets: ' || SQLERRM);

-- You can log the error or perform any necessary error handling here

END generate\_tickets;

END PKG\_BOOKING\_PROCESS;

/