**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 9999999999999999999999999999 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)

);

**Procedures and Functions:**

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;

/

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;

/

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;

/

**Triggers:**

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;

/

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;

/