**CREATE USER SHAMIM IDENTIFIED BY "shamim";**

**GRANT CREATE VIEW, CONNECT, RESOURCE, UNLIMITED TABLESPACE TO SHAMIM;**

**--- Creating a table named 'manager'**

**CREATE TABLE MANAGER (**

**MGR\_ID NUMBER(10) PRIMARY KEY,**

**MGR\_NAME VARCHAR2(20),**

**MGR\_PASS VARCHAR2(20),**

**MGR\_EMAIL VARCHAR2(50),**

**MGR\_PHONE VARCHAR2(20)**

**);**

**-- creating two sequences to manage automatic id generation**

**CREATE SEQUENCE MGR\_SEQ START WITH 1;**

**CREATE SEQUENCE MY\_SEQ START WITH 1;**

**INSERT INTO MANAGER VALUES (**

**MGR\_SEQ.NEXTVAL,**

**'manager',**

**'123',**

**'mgr'**

**||MY\_SEQ.NEXTVAL**

**||'@gmail.com',**

**'0123404239'**

**);**

**-- Creating a Table named 'passenger'**

**CREATE TABLE PASSENGER (**

**PASS\_ID NUMBER(10) PRIMARY KEY,**

**PASS\_NAME VARCHAR2(20),**

**PASS\_PASSWORD VARCHAR2(20),**

**PASS\_EMAIL VARCHAR2(50),**

**PASS\_PHONE VARCHAR2(20),**

**MGR\_ID NUMBER(10),**

**FOREIGN KEY (MGR\_ID) REFERENCES MANAGER (MGR\_ID)**

**);**

**-- creating a sequence which will start from 1 named 'pass\_seq'**

**CREATE SEQUENCE PASS\_SEQ START WITH 1;**

**INSERT INTO PASSENGER VALUES (**

**PASS\_SEQ.NEXTVAL,**

**'user'**

**||MY\_SEQ.NEXTVAL,**

**MY\_SEQ.NEXTVAL,**

**'passenger'**

**||MY\_SEQ.NEXTVAL**

**||'@gmail.com',**

**'123123123',**

**MY\_SEQ.NEXTVAL-11**

**);**

**-- creating a table named ticket**

**CREATE TABLE TICKET (**

**TICKET\_ID NUMBER(10) PRIMARY KEY,**

**TOTAL\_TICKET NUMBER(1),**

**TICKET\_STATUS VARCHAR2(10),**

**PASS\_ID NUMBER(10),**

**FOREIGN KEY (PASS\_ID) REFERENCES PASSENGER (PASS\_ID)**

**);**

**CREATE SEQUENCE TICKET\_SEQ START WITH 1;**

**INSERT INTO TICKET VALUES (**

**TICKET\_SEQ.NEXTVAL,**

**2,**

**'Booked',**

**1**

**);**

**-- creating a table named schedule**

**CREATE TABLE SCHEDULE (**

**SCH\_ID NUMBER(10) PRIMARY KEY,**

**DEPARTURE VARCHAR2(20),**

**DESTINATION VARCHAR2(20),**

**DEPARTURE\_TIME DATE,**

**ARRIVAL\_TIME DATE,**

**COST NUMBER(10, 2),**

**MGR\_ID NUMBER(10),**

**FOREIGN KEY (MGR\_ID) REFERENCES MANAGER (MGR\_ID)**

**);**

**CREATE SEQUENCE SCHE\_SEQ START WITH 1;**

**INSERT INTO SCHEDULE VALUES (**

**SCHE\_SEQ.NEXTVAL,**

**'Dhaka',**

**'Khulna',**

**TO\_DATE('27-05-23 12:00 p.m.', 'dd-mm-yy hh:mi a.m.'),**

**TO\_DATE('27-05-23 7:30 p.m.', 'dd-mm-yy hh:mi a.m.'),**

**960,**

**2**

**);**

**-- booking table**

**CREATE TABLE BOOK (**

**BOOK\_ID NUMBER(10) PRIMARY KEY,**

**PASS\_ID NUMBER(10),**

**SCH\_ID NUMBER(10),**

**FOREIGN KEY (PASS\_ID) REFERENCES PASSENGER (PASS\_ID),**

**FOREIGN KEY (SCH\_ID) REFERENCES SCHEDULE (SCH\_ID)**

**);**

**CREATE SEQUENCE BOOK\_SEQ START WITH 1;**

**INSERT INTO BOOK VALUES (**

**BOOK\_SEQ.NEXTVAL,**

**12,**

**1**

**);**

**-- order table**

**CREATE TABLE ORDERS (**

**ORDER\_ID NUMBER(10) PRIMARY KEY,**

**TICKET\_ID NUMBER(10),**

**SCHE\_ID NUMBER(10),**

**FOREIGN KEY (TICKET\_ID) REFERENCES TICKET (TICKET\_ID),**

**FOREIGN KEY (SCHE\_ID) REFERENCES SCHEDULE (SCH\_ID)**

**);**

**CREATE SEQUENCE ORDER\_SEQ START WITH 1;**

**INSERT INTO ORDERS VALUES (**

**ORDER\_SEQ.NEXTVAL,**

**2,**

**1**

**);**

**-- train\_class table**

**CREATE TABLE TRAIN\_CLASS (**

**CLASS\_ID NUMBER(10) PRIMARY KEY,**

**CLASS VARCHAR2(20),**

**MIN\_COST NUMBER(10),**

**MAX\_COST NUMBER(10)**

**);**

**CREATE SEQUENCE TRAIN\_CLASS\_SEQ START WITH 1;**

**INSERT INTO TRAIN\_CLASS VALUES (**

**TRAIN\_CLASS\_SEQ.NEXTVAL,**

**'First Class',**

**6001,**

**9999**

**);**

**-- #Searching & Advanced Searching Queries**

**-- 1. Search the passenger who's name contains with "%user%"**

**CREATE OR REPLACE VIEW SEARCH\_PASSENGER AS**

**SELECT**

**\***

**FROM**

**PASSENGER**

**WHERE**

**PASS\_NAME LIKE '%user%';**

**SELECT**

**\***

**FROM**

**SEARCH\_PASSENGER;**

**-- 2. Find the total number of tickets sold for each schedule, along with the departure and destination for the schedule**

**SELECT**

**DEPARTURE,**

**DESTINATION,**

**SUM(TOTAL\_TICKET)**

**FROM**

**SCHEDULE S,**

**TICKET   T,**

**BOOK     B**

**WHERE**

**S.SCH\_ID = B.SCH\_ID**

**GROUP BY**

**DEPARTURE,**

**DESTINATION;**

**-- 3. Find the number of manager based on their salary**

**SELECT**

**MGR\_SALARY,**

**COUNT(\*)**

**FROM**

**MANAGER**

**GROUP BY**

**MGR\_SALARY;**

**-- 4. Retrieve the passenger ID, name, email, manager name, and manager email for any specific passenger by his/her name from the passenger and manager tables.**

**SELECT**

**P.PASS\_ID,**

**P.PASS\_NAME,**

**P.PASS\_EMAIL,**

**M.MGR\_NAME,**

**M.MGR\_EMAIL**

**FROM**

**PASSENGER P**

**JOIN MANAGER M**

**ON P.MGR\_ID = M.MGR\_ID**

**WHERE**

**P.PASS\_NAME = 'Shamim'**

**-- 5. Find of all managers who have managed a schedule with a cost greater than $1000**

**SELECT**

**MGR\_NAME,**

**SUM(S.COST)**

**FROM**

**MANAGER  M,**

**SCHEDULE S**

**WHERE**

**M.MGR\_ID = S.MGR\_ID**

**AND S.COST > 1000**

**GROUP BY**

**MGR\_NAME,**

**MGR\_EMAIL,**

**MGR\_PHONE;**

**-- 6. Find the total number of tickets sold for each schedule, along with the departure and destination, where the tickets are either "Active" or "Booked" status and schedules are managed by a manager with an ID of 10, and the departure time is after 6 PM**

**SELECT**

**DEPARTURE,**

**DESTINATION,**

**SUM(TOTAL\_TICKET)**

**FROM**

**SCHEDULE S,**

**TICKET   T,**

**BOOK     B**

**WHERE**

**S.SCH\_ID = B.SCH\_ID**

**AND T.TICKET\_STATUS IN ('Active',**

**'Booked')**

**AND S.MGR\_ID = 10**

**AND S.DEPARTURE\_TIME > '18:00:00'**

**AND (T.TOTAL\_TICKET \* S.COST)>500**

**GROUP BY**

**DEPARTURE,**

**DESTINATION;**

**-- #Procedures**

**-- 1. A user registration package that includes a procedure, function, and proper exception handling.**

**CREATE OR REPLACE PACKAGE PKG\_USER\_REGISTRATION IS**

**PROCEDURE REGISTER(**

**P\_NAME IN PASSENGER.PASS\_NAME%TYPE,**

**P\_PASSWORD IN PASSENGER.PASS\_PASSWORD%TYPE,**

**P\_EMAIL IN PASSENGER.PASS\_EMAIL%TYPE,**

**P\_PHONE IN PASSENGER.PASS\_PHONE%TYPE,**

**P\_MGR\_ID IN PASSENGER.MGR\_ID%TYPE**

**);**

**END PKG\_USER\_REGISTRATION;**

**CREATE OR REPLACE PACKAGE BODY PKG\_USER\_REGISTRATION IS**

**FUNCTION IS\_PASSENGER\_UNIQUE(**

**P\_EMAIL PASSENGER.PASS\_EMAIL%TYPE**

**) RETURN BOOLEAN IS**

**V\_COUNT NUMBER := 0;**

**BEGIN**

**SELECT**

**COUNT(\*) INTO V\_COUNT**

**FROM**

**PASSENGER**

**WHERE**

**PASS\_EMAIL = P\_EMAIL;**

**IF V\_COUNT = 0 THEN**

**RETURN TRUE;**

**ELSE**

**RETURN FALSE;**

**END IF;**

**END;**

**PROCEDURE REGISTER(**

**P\_NAME IN PASSENGER.PASS\_NAME%TYPE,**

**P\_PASSWORD IN PASSENGER.PASS\_PASSWORD%TYPE,**

**P\_EMAIL IN PASSENGER.PASS\_EMAIL%TYPE,**

**P\_PHONE IN PASSENGER.PASS\_PHONE%TYPE,**

**P\_MGR\_ID IN PASSENGER.MGR\_ID%TYPE**

**) IS**

**IS\_PASS\_UNIQUE     BOOLEAN;**

**EXC\_DUPLICATE\_USER EXCEPTION;**

**BEGIN**

**IS\_PASS\_UNIQUE := IS\_PASSENGER\_UNIQUE(P\_EMAIL);**

**IF IS\_PASS\_UNIQUE = FALSE THEN**

**RAISE EXC\_DUPLICATE\_USER;**

**END IF;**

**INSERT INTO PASSENGER VALUES (**

**PASS\_SEQ.NEXTVAL,**

**P\_NAME,**

**P\_PASSWORD,**

**P\_EMAIL,**

**P\_PHONE,**

**P\_MGR\_ID**

**);**

**DBMS\_OUTPUT.PUT\_LINE('The user has been registered successfully.');**

**EXCEPTION**

**WHEN EXC\_DUPLICATE\_USER THEN**

**DBMS\_OUTPUT.PUT\_LINE('The username or email address is not unique. Please try it once again.');**

**END;**

**END PKG\_USER\_REGISTRATION;**

**BEGIN**

**PKG\_USER\_REGISTRATION.REGISTER('Dopa mine', '123123', 'dopa@gmail.com', '01212312334', 1);**

**END;**

**-- 2. A package including procedure, function, and proper exception handling to remove a user based on his email address.**

**CREATE OR REPLACE PACKAGE PKG\_REMOVE\_USER IS**

**PROCEDURE REMOVE(**

**P\_EMAIL IN PASSENGER.PASS\_EMAIL%TYPE**

**);**

**END PKG\_REMOVE\_USER;**

**CREATE OR REPLACE PACKAGE BODY PKG\_REMOVE\_USER IS**

**FUNCTION IS\_PASSENGER\_EXIST(**

**P\_EMAIL PASSENGER.PASS\_EMAIL%TYPE**

**) RETURN BOOLEAN IS**

**V\_COUNT NUMBER := 0;**

**BEGIN**

**SELECT**

**COUNT(\*) INTO V\_COUNT**

**FROM**

**PASSENGER**

**WHERE**

**PASS\_EMAIL = P\_EMAIL;**

**IF V\_COUNT = 0 THEN**

**RETURN FALSE;**

**ELSE**

**RETURN TRUE;**

**END IF;**

**END;**

**PROCEDURE REMOVE(**

**P\_EMAIL IN PASSENGER.PASS\_EMAIL%TYPE**

**) IS**

**IS\_USER\_EXIST      BOOLEAN;**

**EXC\_USER\_NOT\_EXIST EXCEPTION;**

**BEGIN**

**IS\_USER\_EXIST := IS\_PASSENGER\_EXIST(P\_EMAIL);**

**IF IS\_USER\_EXIST = FALSE THEN**

**RAISE EXC\_USER\_NOT\_EXIST;**

**END IF;**

**DELETE FROM PASSENGER**

**WHERE**

**PASS\_EMAIL = P\_EMAIL;**

**DBMS\_OUTPUT.PUT\_LINE('The user has been deleted successfully');**

**EXCEPTION**

**WHEN EXC\_USER\_NOT\_EXIST THEN**

**DBMS\_OUTPUT.PUT\_LINE('The email address is not exist. Please try it once again.');**

**END;**

**END PKG\_REMOVE\_USER;**

**BEGIN**

**PKG\_REMOVE\_USER.REMOVE('dopa@gmail.com');**

**END;**

**-- 3. A package that includes a procedure, function, and proper exception handling to delay all schedules of any specific destination by one day.**

**CREATE OR REPLACE PACKAGE PKG\_DELAY\_SCHEDULE IS**

**PROCEDURE DELAY(**

**P\_DESTINATION IN SCHEDULE.DESTINATION%TYPE**

**);**

**END PKG\_DELAY\_SCHEDULE;**

**CREATE OR REPLACE PACKAGE BODY PKG\_DELAY\_SCHEDULE IS**

**FUNCTION IS\_SCHEDULE\_EXIST(**

**P\_DESTINATION SCHEDULE.DESTINATION%TYPE**

**) RETURN BOOLEAN IS**

**V\_COUNT NUMBER := 0;**

**BEGIN**

**SELECT**

**COUNT(\*) INTO V\_COUNT**

**FROM**

**SCHEDULE**

**WHERE**

**DESTINATION = P\_DESTINATION;**

**IF V\_COUNT = 0 THEN**

**RETURN FALSE;**

**ELSE**

**RETURN TRUE;**

**END IF;**

**END;**

**PROCEDURE DELAY(**

**P\_DESTINATION IN SCHEDULE.DESTINATION%TYPE**

**) IS**

**IS\_SCH\_EXIST      BOOLEAN;**

**EXC\_SCH\_NOT\_EXIST EXCEPTION;**

**BEGIN**

**IS\_SCH\_EXIST := IS\_SCHEDULE\_EXIST(P\_DESTINATION);**

**IF IS\_SCH\_EXIST = FALSE THEN**

**RAISE EXC\_SCH\_NOT\_EXIST;**

**END IF;**

**UPDATE SCHEDULE**

**SET**

**DEPARTURE\_TIME = DEPARTURE\_TIME + INTERVAL '1' DAY,**

**ARRIVAL\_TIME = ARRIVAL\_TIME + INTERVAL '1' DAY**

**WHERE**

**DESTINATION = P\_DESTINATION;**

**DBMS\_OUTPUT.PUT\_LINE('The schedule(s) has been delayed by 1 day.');**

**EXCEPTION**

**WHEN EXC\_SCH\_NOT\_EXIST THEN**

**DBMS\_OUTPUT.PUT\_LINE('There is no schedule available for that destination.');**

**END;**

**END PKG\_DELAY\_SCHEDULE;**

**BEGIN**

**PKG\_DELAY\_SCHEDULE.DELAY('Chittagong');**

**END;**

**-- 4. A stored procedure to update email address of a passenger**

**CREATE PROCEDURE UPDATE\_PASSENGER\_EMAIL( P\_PASS\_ID IN PASSENGER.PASS\_ID%TYPE, P\_NEW\_EMAIL IN PASSENGER.PASS\_EMAIL%TYPE ) AS**

**BEGIN**

**UPDATE PASSENGER**

**SET**

**PASS\_EMAIL = P\_NEW\_EMAIL**

**WHERE**

**PASS\_ID = P\_PASS\_ID;**

**END;**

**BEGIN**

**UPDATE\_PASSENGER\_EMAIL (5, 'updated@gmail.com');**

**END;**

**-- 5. A stored procedure to update salary of manager using a threshold**

**CREATE OR REPLACE PROCEDURE UPDATESALARY(P\_THRESHOLD IN MANAGER.MGR\_SALARY%TYPE, P\_AMOUNT IN MANAGER.MGR\_SALARY%TYPE) AS**

**CURSOR C\_MANAGER IS**

**SELECT**

**MGR\_ID**

**FROM**

**MANAGER**

**WHERE**

**MGR\_SALARY > P\_THRESHOLD;**

**BEGIN**

**FOR R\_MANAGER IN C\_MANAGER LOOP**

**UPDATE MANAGER**

**SET**

**MGR\_SALARY = MGR\_SALARY + P\_AMOUNT**

**WHERE**

**MGR\_ID = R\_MANAGER.MGR\_ID;**

**END LOOP;**

**END;**

**BEGIN**

**UPDATESALARY(4000, 1000);**

**END;**

**-- 6. A stored procedure to delete all cancelled tickets**

**CREATE PROCEDURE DELETE\_CANCELLED\_TICKETS AS**

**CURSOR C\_TICKET IS**

**SELECT**

**TICKET\_ID**

**FROM**

**TICKET**

**WHERE**

**TICKET\_STATUS = 'Cancelled';**

**BEGIN**

**FOR R\_TICKET IN C\_TICKET LOOP**

**DELETE FROM TICKET**

**WHERE**

**TICKET\_ID = R\_TICKET.TICKET\_ID;**

**END LOOP;**

**END;**

**BEGIN**

**DELETE\_CANCELLED\_TICKETS;**

**END;**

**-- # Triggers**

**-- 1. A trigger to enforce the constraint that each schedule can have at most 50 rows with the same destination and an arrival time that is less than the current date and time.**

**CREATE OR REPLACE TRIGGER TRG\_LIMIT\_DESTINATION BEFORE INSERT OR UPDATE ON SCHEDULE FOR EACH ROW DECLARE V\_COUNT NUMBER;**

**BEGIN**

**SELECT**

**COUNT(\*) INTO V\_COUNT**

**FROM**

**SCHEDULE**

**WHERE**

**DESTINATION = :NEW.DESTINATION**

**AND ARRIVAL\_TIME < SYSDATE;**

**IF V\_COUNT >= 50 THEN**

**RAISE\_APPLICATION\_ERROR(-20001, 'There can be at most 50 rows with the same destination and an arrival time less than the current date and time');**

**END IF;**

**END;**

**-- 2. Create a backup table of passenger with trigger**

**CREATE OR REPLACE TABLE PASSENGER\_BACKUP AS**

**SELECT           \* FROM PASSENGER WHERE 1 = 2;**

**CREATE           OR REPLACE TRIGGER BACKUP\_PASSENGER\_TRIGGER AFTER INSERT OR UPDATE OR DELETE ON PASSENGER FOR EACH ROW BEGIN INSERT INTO PASSENGER\_BACKUP VALUES (**

**:NEW.PASS\_ID,**

**:NEW.PASS\_NAME,**

**:NEW.PASS\_PASSWORD,**

**:NEW.PASS\_EMAIL,**

**:NEW.PASS\_PHONE,**

**:NEW.MGR\_ID**

**);**

**END;**

**-- 3. A trigger to keep log of insert, update, and delete operation on each row of passenger table**

**CREATE           TABLE PASSENGER\_LOG ( LOG\_ID NUMBER PRIMARY KEY, PASS\_ID NUMBER, PASS\_NAME VARCHAR2(20), PASS\_PASSWORD VARCHAR2(20), PASS\_EMAIL VARCHAR2(50), PASS\_PHONE VARCHAR2(20), MGR\_ID NUMBER, LOG\_OPERATION VARCHAR2(10), LOG\_TIMESTAMP TIMESTAMP );**

**CREATE           SEQUENCE PASS\_LOG\_SEQ START WITH 1;**

**CREATE           OR REPLACE TRIGGER PASSENGER\_TRG AFTER INSERT OR DELETE OR UPDATE ON PASSENGER FOR EACH ROW BEGIN IF INSERTING THEN INSERT INTO PASSENGER\_LOG VALUES (**

**PASS\_LOG\_SEQ.NEXTVAL,**

**:NEW.PASS\_ID,**

**:NEW.PASS\_NAME,**

**:NEW.PASS\_PASSWORD,**

**:NEW.PASS\_EMAIL,**

**:NEW.PASS\_PHONE,**

**:NEW.MGR\_ID,**

**'INSERT',**

**SYSTIMESTAMP**

**);**

**ELSIF            DELETING THEN**

**INSERT INTO PASSENGER\_LOG VALUES (**

**PASS\_LOG\_SEQ.NEXTVAL,**

**:OLD.PASS\_ID,**

**:OLD.PASS\_NAME,**

**:OLD.PASS\_PASSWORD,**

**:OLD.PASS\_EMAIL,**

**:OLD.PASS\_PHONE,**

**:OLD.MGR\_ID,**

**'DELETE',**

**SYSTIMESTAMP**

**);**

**ELSIF            UPDATING THEN**

**INSERT INTO PASSENGER\_LOG VALUES (**

**PASS\_LOG\_SEQ.NEXTVAL,**

**:NEW.PASS\_ID,**

**:NEW.PASS\_NAME,**

**:NEW.PASS\_PASSWORD,**

**:NEW.PASS\_EMAIL,**

**:NEW.PASS\_PHONE,**

**:NEW.MGR\_ID,**

**'UPDATE',**

**SYSTIMESTAMP**

**);**

**END IF;**

**END;**

**-- 4. Prevent updating an order to a non-existent ticket or schedule**

**CREATE           OR REPLACE TRIGGER PREVENT\_INVALID\_ORDER\_UPDATE BEFORE UPDATE ON ORDERS FOR EACH ROW DECLARE V\_TICKET\_COUNT NUMBER;**

**V\_SCHEDULE\_COUNT NUMBER;**

**BEGIN**

**SELECT**

**COUNT(\*) INTO V\_TICKET\_COUNT**

**FROM**

**TICKET**

**WHERE**

**TICKET\_ID = :NEW.TICKET\_ID;**

**SELECT**

**COUNT(\*) INTO V\_SCHEDULE\_COUNT**

**FROM**

**SCHEDULE**

**WHERE**

**SCH\_ID = :NEW.SCHE\_ID;**

**IF V\_TICKET\_COUNT = 0 THEN**

**RAISE\_APPLICATION\_ERROR(-20006, 'Invalid Ticket ID');**

**ELSIF V\_SCHEDULE\_COUNT = 0 THEN**

**RAISE\_APPLICATION\_ERROR(-20009, 'Invalid Schedule ID');**

**END IF;**

**END;**