American International University-Bangladesh (AIUB)

Faculty of Science and Technology

Department of Computer Science and Engineering

CSC 4181: Advanced Database Management Systems

Final Term Project Report

Fall 2022-23

Project Name

MetroTicket

Team

Runtime Terror

Member

Zaid Amin Rawfin

20-42459-1

Section: **C**

Table of Contents

Searching & Advanced Searching Queries

```
1. Search the passenger who's name ends with "fin"

CREATE OR REPLACE VIEW search_passenger AS
SELECT *
FROM PASSENGER
WHERE PASS_NAME LIKE '%fin';


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 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;


4. Find the departure and destination for all schedules that have
had at least one order made for them, along with the total number
of orders made for each schedule, where the schedules have a cost
less than $50 and the orders were made by a passenger with an ID
of 10.

SELECT departure, destination, COUNT(order_id)
FROM schedule s, orders o, ticket t
WHERE s.sch_id = o.sche_id AND o.ticket_id = t.ticket_id
  AND s.cost < 50 AND t.pass_id = 10
GROUP BY departure, destination;


5. 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;
```

Sequences

```
1. Sequence for Passenger table
CREATE SEQUENCE pass_seq START WITH 1;

INSERT INTO passenger VALUES (pass_seq.NEXTVAL, 'Raofin',
       '~Raof!n^^', 'raofin@hotmail.com', '012345678', 1);

2. Sequence for Manager table
CREATE SEQUENCE mgr_seq START WITH 1;

INSERT INTO manager VALUES (mgr_seq.NEXTVAL, 'Zaid',
       'zaid123', 'zaid@email.com', '0123649849810');

3. Sequence for Ticket table
CREATE SEQUENCE ticket_seq START WITH 1;

INSERT INTO ticket VALUES (ticket_seq.NEXTVAL, 2, 'Booked', 1);

4. Sequence for Book table
CREATE SEQUENCE sche_seq START WITH 1;

INSERT INTO schedule VALUES (sche_seq.NEXTVAL, 'Dhaka',
       'Noakhali', TO_DATE('07-11-22 11:59 a.m.',
       'dd-mm-yy hh:mi a.m.'), TO_DATE('07-11-22 11:30 a.m.',
       'dd-mm-yy hh:mi a.m.'), 9600, 2);

5. Sequence for Order table
CREATE SEQUENCE book_seq START WITH 1;

INSERT INTO book VALUES (book_seq.NEXTVAL, 1, 1);
```

Views

```
1. Find out train arriving time for each passengers.
CREATE OR REPLACE VIEW train_arriving_each_passenger AS
SELECT pass_name, arrival_time
FROM passenger c, schedule f, book b
WHERE c.pass_id = b.pass_id AND b.sch_id = f.sch_id;


2. Find the total number of tickets booked for each train
schedules.
CREATE OR REPLACE VIEW booked_tickets_each_schedules AS
SELECT schedule.sch_id, SUM(ticket.total_ticket) AS total_tickets
FROM schedule, ticket, orders
WHERE ticket.ticket_id = orders.ticket_id
  AND orders.sche_id = schedule.sch_id
GROUP BY schedule.sch_id;


3. Find the departure time of the cheapest first-class train
schedule.
CREATE OR REPLACE VIEW cheapest_first_class_trail AS
SELECT * FROM schedule, train_class
WHERE cost BETWEEN min_cost AND max_cost AND cost IN
  (SELECT MIN(cost) FROM schedule, train_class WHERE cost
   BETWEEN min_cost AND max_cost AND class = 'First Class');


4. A view that displays the names, email addresses, and phone
numbers of all managers who have managed a schedule with a cost
greater than $500
CREATE VIEW high_cost_schedules AS
SELECT mgr_name, mgr_email, mgr_phone
FROM manager m, schedule s
WHERE m.mgr_id = s.mgr_id AND s.cost > 500
GROUP BY mgr_name, mgr_email, mgr_phone;


5. Find the manager who managed the maximum train schedules.
CREATE OR REPLACE VIEW managed_max_schedules AS
SELECT * FROM manager WHERE mgr_id IN
   (SELECT m.mgr_id FROM manager m, schedule s
    WHERE m.mgr_id = s.mgr_id GROUP BY m.mgr_id
    HAVING COUNT(m.mgr_id) IN
        (SELECT MAX(COUNT(m.mgr_id)) FROM manager m, schedule s
         WHERE m.mgr_id = s.mgr_id
         GROUP BY m.mgr_id));
```

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_name passenger.pass_name%TYPE, p_email passenger.pass_email%TYPE)
        RETURN BOOLEAN IS
        v_count NUMBER := 0;
    BEGIN
        SELECT COUNT(*) INTO v_count FROM passenger
        WHERE pass_name = p_name OR 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_name, 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('Rawfin', '#wf$', 'fin@email.com', '01234', 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 unccessfully');

        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('rawfin@email.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('Noakhali');
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 (2, 'new_email@example.com');
END;
```

5. A stored procedure to update the phone number of all passengers in a specified manager's group

```
CREATE PROCEDURE update_phone_for_manager_group(
              p_mgr_id IN passenger.mgr_id%TYPE,
              p_new_phone IN passenger.pass_phone%TYPE
) AS
    CURSOR c_passenger IS
        SELECT pass_id FROM passenger WHERE mgr_id = p_mgr_id;
BEGIN
    FOR r_passenger IN c_passenger
        LOOP
            UPDATE passenger SET pass_phone = p_new_phone
            WHERE pass_id = r_passenger.pass_id;
        END LOOP;
END;

BEGIN
    update_phone_for_manager_group(1, '9876543210');
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 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;
```