TEAM – 05

(ADBM – PL/SQL) reference

Team members :

- M S F Safra – GAHDSE241F-010
- N P Dissanayaka – GAHDSE241F-022
- C D Wijesekara – GAHDSE241F-045

--DDL COMMANDS

1. **Table Creation:**

Flight Table:

```
create table flight(
  FCode varchar2(10) primary key,
  FType varchar2(20) NOT NULL,
  Departure varchar2(50),
  Arrival varchar2(50),
  DDate date,
  ADate date,
  constraint chk_FCode CHECK (FCode LIKE 'FL%')
);
```

- Explanation:
  - FCode: A primary key that uniquely identifies a flight and must start with 'FL'.
  - FType: The flight type (e.g., Boeing, Commercial).
  - Departure and Arrival: Departure and arrival cities.
  - DDate and ADate: Departure and arrival dates.
  - The table ensures that FCode starts with 'FL' using a CHECK constraint.

Maintenance Table:

```
create table maintenance(
  MID number(5) primary key,
  MDate date not null,
  MCost Number(10) not null,
  FCode varchar2(10),
  constraint fk_FCode foreign key(FCode) references flight (FCode)
);
```

- Explanation:
    - MID: Maintenance ID, a primary key.
    - MDate: Maintenance date.
    - MCost: Maintenance cost.
    - FCode: Foreign key that references the flight table, linking maintenance records to flights.

Airport Table:

```
create table airport(
  APNo varchar2(10) primary key,
  APName varchar(50) not null,
  city varchar(50) not null
);
```

- Explanation:
    - APNo: Airport number (primary key).
    - APName: Name of the airport.
    - city: City where the airport is located.

Travels Table:

```
create table travels(
  FCode varchar2(10),
  APNo varchar2(10),
  constraint pk_travels primary key(FCode, APNo),
  constraint fk_travels_FCode foreign key (FCode) references flight(FCode),
  constraint fk_travels_APNo foreign key (APNo) references airport(APNo)
);
```

- Explanation:
    - This table represents the relationship between flights (FCode) and airports (APNo), with both as foreign keys.

Employee and Crew/Admin Tables:

```
create table employee(
  EID Number primary key,
  EName varchar2(50) not null,
  FCode varchar(10) not null,
  constraint fk_employee_FCode foreign key (FCode) references flight(FCode)
);
```

- Explanation:
  - EID: Employee ID (primary key).
  - EName: Employee name.
  - FCode: The flight they are assigned to (foreign key to the flight table).

employeeCrew and employeeAdmin tables store specific employee roles as crew or admin by referencing the EID of the employee table.

---

2. **Sequences:**

```
create sequence maintenance_seq
start with 1
increment by 1
nocache;
```

- Explanation:
  - A sequence named maintenance_seq is created to automatically generate unique maintenance IDs (MID).
  - Similarly, sequences for employee and passenger tables generate unique IDs for each record.

---

3. **Insert Statements:**

```
INSERT INTO flight (
  FCode, FType, Departure, Arrival, DDate, ADate
```

```
)
VALUES (
  'FL001', 'Boeing', 'Dubai', 'Colombo',
  TO_DATE('2024-09-30', 'YYYY-MM-DD'),
  TO_DATE('2024-10-01', 'YYYY-MM-DD')
);
```

- Explanation:
  - This inserts a flight record into the flight table with specific values for FCode, FType, departure/arrival cities, and dates.

---

4. **Joins:**

Inner Join:

```
select f.departure, f.FType, e.eid, e.ename
from flight f
inner join employee e on f.fcode = e.fcode;
```

- Explanation:
  - An inner join retrieves all records from flight and employee where there is a matching FCode in both tables.

Left Join:

```
select f.departure, f.FType, e.eid, e.ename
from flight f
left join employee e on f.fcode = e.fcode;
```

- Explanation:
  - A left join retrieves all records from flight, even if there are no matching records in employee.

Full Outer Join:

```
select * from flight f
full join employee e on f.fcode = e.fcode;
```

- Explanation:

- A full outer join retrieves all records from both flight and employee, showing unmatched records from both tables where necessary.

---

5. **Update and Delete Statements:**

Update Flight:

```
begin
  update flight
  set departure = 'Los Angeles'
  where FCode = 'FL001';
end;
```

- Explanation:
  - Updates the departure city of flight FL001 to "Los Angeles".

Delete Flight:

```
begin
  delete from flight
  where FCode = 'FL123';
end;
```

- Explanation:
  - Deletes the flight record with FCode = 'FL123' from the flight table.

---

6. **Group By and Aggregation:**

```
SELECT departure, MAX(FType) AS FType
FROM flight
GROUP BY departure;
```

- Explanation:
  - Retrieves the maximum flight type for each departure city.

---

7. **Cursor and Loop Example:**

```
begin
  for rec in (
    select departure, MAX(FType) As FType
    from flight
    where departure = 'Los Angeles'
    group by departure
    order by departure asc
  )
  loop
    dbms_output.put_line('Departure: ' || rec.departure || ', FType: ' ||
rec.FType);
  end loop;
end;
```

- Explanation:
    - A FOR loop that selects and prints the maximum FType for flights departing from Los Angeles.

Conclusion:

This script covers:

- Table Creation: Defining schemas and relationships between entities like flights, employees, maintenance, etc.
- Sequences: Auto-generating IDs for tables.
- Data Manipulation: Inserting, updating, deleting records, and performing various queries using joins and aggregation.
- Joins: Combining data across related tables (inner, left, full joins).
- PL/SQL: Using blocks for transactions, handling loops, and managing outputs.

PL/ SQL

1. **Single Row Insert with User Input:**

```
set serveroutput on;
declare
  v_departure varchar2(50);
  v_arrival varchar2(50);
begin
  v_departure := '&departure';  -- Accepts user input for departure city.
  v_arrival := '&arrival';     -- Accepts user input for arrival city.

  -- Inserts a flight record into the 'flight' table.
  insert into flight(FCode, FType, Departure, Arrival, DDate, ADate)
  values('FL006', 'Boeing', v_departure, v_arrival, SYSDATE, SYSDATE + 1);

  dbms_output.put_line('Flight details inserted successfully.');
end;
/
```

- Explanation:
  - v_departure and v_arrival are variables to store user inputs (departure and arrival locations).
  - SYSDATE is a system function that returns the current date and time.
  - SYSDATE + 1 means 1 day after the current date.
  - The code inserts a flight record with a fixed flight code (FL006), flight type (Boeing), and the user-specified departure and arrival cities, with the departure date as the current date and arrival date as the next day.
  - After the insert, it prints a success message.

---

2. **Multiple Row Insert Using Arrays and Conditional Logic:**

```
DECLARE
```

```
  -- Define an array (collection) for departure and arrival
  TYPE location_array IS TABLE OF VARCHAR2(50) INDEX BY
PLS_INTEGER;
  v_departure location_array;
  v_arrival location_array;
BEGIN
  -- Initialize array with values
  v_departure(1) := 'New York';
  v_arrival(1) := 'Los Angeles';

  v_departure(2) := 'Chicago';
  v_arrival(2) := 'Miami';

  -- Loop through the array to insert multiple records
  FOR i IN 1 .. 2 LOOP
    -- Use IF-ELSE logic for a simple condition
    IF v_departure(i) = 'Chicago' THEN
      INSERT INTO flight (FCode, FType, Departure, Arrival, DDate, ADate)
      VALUES ('FL007', 'Airbus', v_departure(i), v_arrival(i), SYSDATE,
SYSDATE + 1);
    ELSE
      INSERT INTO flight (FCode, FType, Departure, Arrival, DDate, ADate)
      VALUES ('FL008', 'Boeing', v_departure(i), v_arrival(i), SYSDATE,
SYSDATE + 1);
    END IF;
  END LOOP;

  DBMS_OUTPUT.PUT_LINE('Flight details inserted successfully.');
END;
```

/

- Explanation:
  - The code defines two arrays, v_departure and v_arrival, which hold departure and arrival city values.
  - Two sets of values are added to these arrays:
    - New York → Los Angeles
    - Chicago → Miami
  - The code then loops through the array indices (1 and 2) to insert two flight records:
    - If the departure is "Chicago", it inserts a flight with FCode as FL007 and FType as Airbus.
    - For other cities, it inserts a flight with FCode as FL008 and FType as Boeing.
  - The insertion includes the current date (SYSDATE) as the departure date and the next day (SYSDATE + 1) as the arrival date.
  - A message confirms the insertion.

---

3. Cursor for Fetching and Displaying Flight Data:

declare

cursor flight_cursor is

select FCode, Departure from flight; -- Cursor selects flight code and departure city.

v_fcode flight.FCode%TYPE;

v_departure flight.Departure%TYPE;

begin

open flight_cursor; -- Open the cursor to fetch rows.

loop

```
    fetch flight_cursor into v_fcode, v_departure;  -- Fetch the next row.
    exit when flight_cursor%NOTFOUND;  -- Exit loop when no more rows.

    dbms_output.put_line('Flight Code: '|| v_fcode || ', Departure: '||
v_departure);
  end loop;

  close flight_cursor;  -- Close the cursor after use.
end;
/
```

- Explanation:
  - A cursor flight_cursor is declared to fetch flight codes (FCode) and departure cities (Departure) from the flight table.
  - The cursor is opened, and the results are fetched row by row into the variables v_fcode and v_departure.
  - Inside a loop, each flight's code and departure city are printed.
  - The loop continues until all rows are fetched (flight_cursor%NOTFOUND), and then the cursor is closed.

---

4. Insert with Exception Handling:

```
begin
  -- Attempt to insert a flight record into the 'flight' table.
  insert into flight(FCode, FType, Departure, Arrival, DDate, ADate)
  values('FL003','Boeing','Toronto','Saint Petersburgh', SYSDATE,SYSDATE
+1);

  dbms_output.put_line('Flight details inserted successfully.');

exception
```

```
    when DUP_VAL_ON_INDEX then
      dbms_output.put_line('Error: Flight code already exists');
    when OTHERS then
      dbms_output.put_line('An error occurred: '|| SQLERRM);
  end;
  /
```

- Explanation:
  - The code attempts to insert a new flight record with FCode as FL003, flight type Boeing, and cities "Toronto" and "Saint Petersburgh".
  - If the flight code FL003 already exists, the DUP_VAL_ON_INDEX exception is triggered (i.e., a duplicate value error on a unique index), and an error message is printed.
  - The OTHERS clause is a catch-all for any other exceptions that might occur during the execution, and the error message is printed using SQLERRM, which provides the error details.
  - If no exceptions are raised, a success message is printed.

Summary of Key Concepts:

- Variables: Used to store inputs or dynamically assigned values.
- Arrays (Collections): Store multiple values and allow iteration for bulk operations.
- Loops: Iterates over arrays or results of a query to process multiple records.
- Cursors: Allows you to fetch and process query results row by row.
- Conditions: IF-ELSE logic enables conditional data processing during inserts.
- Exception Handling: Handles runtime errors gracefully, such as duplicate entries or other unforeseen issues.