

DMDD Phase 3

Team 16

Step 1: Create MBTA_Admin user using oracle admin

File Name: Roles/Mbta_Admin_User_Creation_1.sql

Worksheet | Query Builder

```
/*
  This is done by ADMIN USER to create an mbta_admin for the MBTA project
*/

SET SERVEROUTPUT ON;

DECLARE
    user_exists NUMBER;
    grant_error EXCEPTION;
BEGIN
    -- Check if the user already exists
    SELECT COUNT(*)
    INTO user_exists
    FROM all_users
    WHERE username = 'MBTA_ADMIN';

    IF user_exists = 0 THEN
        -- Create the user if it doesn't exist
        EXECUTE IMMEDIATE 'CREATE USER mbta_admin IDENTIFIED BY "AdminRole@1234"';
        DBMS_OUTPUT.PUT_LINE('User "mbta_admin" created successfully.');
    ELSE
        -- User already exists, inform the user
        DBMS_OUTPUT.PUT_LINE('User "mbta_admin" already exists.');
    END IF;

    BEGIN
        -- Grant necessary privileges to mbta_admin
    END;
END;
```

Script Output x

Task completed in 1.093 seconds

User "mbta_admin" already exists.
All necessary privileges granted to "mbta_admin".

PL/SQL procedure successfully completed.

Login to Mbta_admin and then start creating tables

Step 2(DDL) : Create Tables in the order mentioned using MBTA_Admin to avoid foreign Key constraints

- Discount Table script: DDL/Create_Discounts_Table.sql

SQL Worksheet History

Worksheet Query Builder

```

SET SERVEROUTPUT ON;
-- Create DISCOUNTS table if it does not already exist using mbta admin
DECLARE
    table_count INTEGER;
BEGIN
    -- Check if the DISCOUNTS table exists
    SELECT COUNT(*) INTO table_count FROM all_tables WHERE table_name = 'DISCOUNTS';

    IF table_count = 0 THEN
        EXECUTE IMMEDIATE '
            CREATE TABLE discounts (
                discount_id_pk INTEGER PRIMARY KEY,      -- Primary key for discounts
                discount_type VARCHAR2(50),               -- Type of discount (e.g., Student, Senior)
                discount_rate NUMBER(5,2)                 -- Discount rate as a percentage
            );
        ';
        DBMS_OUTPUT.PUT_LINE('Table "DISCOUNTS" created successfully.');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Table "DISCOUNTS" already exists. Skipping creation...');
    END IF;
END;
/

```

Script Output

Task completed in 0.574 seconds

Table "DISCOUNTS" already exists. Skipping creation...

PL/SQL procedure successfully completed.

- User Table Script: DDL/Create_User_Table.sql as mbta admin

Oracle Connections

- Admin
- cashier_user
- customer_user
- dev
- group_booking_manager_user
- HR
- mbta_admin
- subscription_manager_user
- Database Schema Service Connections

Worksheet Query Builder

```

SET SERVEROUTPUT ON;
-- Create USER_TABLE if it does not already exist using mbta_admin
DECLARE
    table_count INTEGER;
BEGIN
    -- Check if the table exists
    SELECT COUNT(*) INTO table_count FROM all_tables WHERE table_name = 'USER_TABLE';

    IF table_count = 0 THEN
        EXECUTE IMMEDIATE '
            CREATE TABLE user_table (
                user_id_pk INTEGER PRIMARY KEY,          -- Primary key for users
                username VARCHAR2(55),                   -- Username of the user
                password VARCHAR2(55),                  -- Password of the user
                email_uniq VARCHAR2(100) UNIQUE,         -- Unique email of the user
                user_category VARCHAR2(55),              -- Category of the user (e.g., Admin, Customer)
                created_at TIMESTAMP,                   -- Date the user was created
                discount_id_fk INTEGER,                -- Foreign Key Reference to discounts table (to be added later)
                CONSTRAINT discount_id_fk FOREIGN KEY (discount_id_fk) REFERENCES discounts(discount_id_pk)
            );
        ';
        DBMS_OUTPUT.PUT_LINE('Table "USER_TABLE" created successfully.');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Table "USER_TABLE" already exists. Skipping creation...');
    END IF;
END;
/

```

Script Output

Task completed in 0.676 seconds

Table "USER_TABLE" already exists. Skipping creation...

PL/SQL procedure successfully completed.

- Group Booking table script: DDL/Create_Group_Booking_Table.sql as mbta admin

```

SET SERVEROUTPUT ON;
-- Create GROUP_BOOKING table if it does not already exist using mbta admin
DECLARE
    table_count INTEGER;
BEGIN
    -- Check if the table exists
    SELECT COUNT(*) INTO table_count FROM all_tables WHERE table_name = 'GROUP_BOOKING';

    IF table_count = 0 THEN
        EXECUTE IMMEDIATE '
        CREATE TABLE group_booking(
            group_booking_id_pk INTEGER PRIMARY KEY, -- Primary key for group bookings
            user_id_fk INTEGER, -- Foreign key reference to user_table
            num_passenger INTEGER, -- Number of passengers in the group
            booking_date DATE, -- Date of the booking
            transit_line VARCHAR2(50), -- Name of the transit line for group booking
            transaction_status VARCHAR2(55), -- Status of the transaction (e.g., Approved, Denied)
            discounts.discount_id_pk INTEGER, -- Foreign key reference to discounts table
            CONSTRAINT fk_user_id_group_booking FOREIGN KEY (user_id_fk) REFERENCES user_table(user_id_pk),
            CONSTRAINT fk_discount_id_group_booking FOREIGN KEY (discounts.discount_id_pk) REFERENCES discounts(discount_id_pk)
        )';
        DBMS_OUTPUT.PUT_LINE('Table "GROUP_BOOKING" created successfully.');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Table "GROUP_BOOKING" already exists. Skipping creation...');
    END IF;
END;
/

```

Script Output:

Task completed in 1.374 seconds
Table "GROUP_BOOKING" already exists. Skipping creation...
PL/SQL procedure successfully completed.

- Rides Table script: DDL/ Create_Rides_Table.sql using mbta_admin

```

SET SERVEROUTPUT ON;
-- Create RIDES table if it does not already exist
DECLARE
    table_count INTEGER;
BEGIN
    -- Check if the table exists
    SELECT COUNT(*) INTO table_count FROM all_tables WHERE table_name = 'RIDES';

    IF table_count = 0 THEN
        EXECUTE IMMEDIATE '
        CREATE TABLE rides(
            ride_id_pk INTEGER PRIMARY KEY, -- Primary key for rides
            user_id_fk INTEGER, -- Foreign key reference to user_table
            total_spent INTEGER, -- Total amount spent on the ride
            CONSTRAINT fk_user_id_rides FOREIGN KEY (user_id_fk) REFERENCES user_table(user_id_pk)
        )';
        DBMS_OUTPUT.PUT_LINE('Table "RIDES" created successfully.');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Table "RIDES" already exists. Skipping creation...');
    END IF;
END;
/

```

Script Output:

Task completed in 6.111 seconds
Table "RIDES" already exists. Skipping creation...
PL/SQL procedure successfully completed.

- subscription table: DDL/ Create_Subscription_Table.sql using mbta_admin

Oracle Connections

- Admin
- cashier_user
- customer_user
- dev
- group_booking_manager_user
- HR
- mbta_admin
- subscription_manager_user

Database Schema Service Connections

Worksheet | Query Builder

```

SET SERVEROUTPUT ON;
-- Create SUBSCRIPTION table if it does not already exist using mbta admin
DECLARE
    table_count INTEGER;
BEGIN
    -- Check if the table exists
    SELECT COUNT(*) INTO table_count FROM all_tables WHERE table_name = 'SUBSCRIPTION';

    IF table_count = 0 THEN
        EXECUTE IMMEDIATE '
            CREATE TABLE subscription (
                subscription_id_pk INTEGER PRIMARY KEY, -- Primary key for subscriptions
                user_id_fk INTEGER, -- Foreign key reference to user_table
                subscription_type VARCHAR2(50), -- Type of subscription (e.g., Monthly, Yearly)
                start_date DATE, -- Start date of the subscription
                end_date DATE, -- End date of the subscription
                subscription_status VARCHAR2(50), -- Status of the subscription (e.g., Active, Expired)
                CONSTRAINT fk_user_id_subscription FOREIGN KEY (user_id_fk) REFERENCES user_table(user_id_pk)
            )';
        DBMS_OUTPUT.PUT_LINE('Table "SUBSCRIPTION" created successfully.');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Table "SUBSCRIPTION" already exists. Skipping creation...');
    END IF;
END;
/

```

Script Output

Task completed in 0.952 seconds

Table "SUBSCRIPTION" already exists. Skipping creation...

PL/SQL procedure successfully completed.

- DDL/Create_Ticket_Table.sql using mbta_admin

Connections | Reports

Oracle Connections

- Admin
- cashier_user
- customer_user
- dev
- group_booking_manager_user
- HR
- mbta_admin
- subscription_manager_user

Database Schema Service Connections

SQL Worksheet: History

Worksheet | Query Builder

```

SET SERVEROUTPUT ON;
-- Create TICKET table if it does not already exist
DECLARE
    table_count INTEGER;
BEGIN
    -- Check if the table exists
    SELECT COUNT(*) INTO table_count FROM all_tables WHERE table_name = 'TICKET';

    IF table_count = 0 THEN
        EXECUTE IMMEDIATE '
            CREATE TABLE ticket (
                ticket_id_pk INTEGER PRIMARY KEY, -- Primary key for tickets
                user_id_fk INTEGER, -- Foreign key reference to user_table
                purchase_date DATE, -- Date of ticket purchase
                price NUMBER(5,2), -- Price of the ticket with 2 decimal places
                transit_line VARCHAR2(50), -- Name of the transit line
                ticket_status VARCHAR2(50), -- Status of the ticket (e.g., Valid, Cancelled)
                CONSTRAINT fk_user_id_ticket FOREIGN KEY (user_id_fk) REFERENCES user_table(user_id_pk)
            )';
        DBMS_OUTPUT.PUT_LINE('Table "TICKET" created successfully.');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Table "TICKET" already exists. Skipping creation...');
    END IF;
END;
/

```

Script Output

Task completed in 1.83099997 seconds

Table "TICKET" already exists. Skipping creation...

PL/SQL procedure successfully completed.

- DDL/ Create_Transaction_Table.sql using mbta_admin

The screenshot shows the Oracle SQL Developer interface. On the left, the 'Oracle Connections' tree view lists several users: Admin, cashier_user, customer_user, dev, group_booking_manager_user, HR, mbta_admin, and subscription_manager_user. The 'Worksheet' tab is active, displaying a PL/SQL script. The code creates a table named 'TRANSACTION_TABLE' if it does not already exist. It includes columns for transaction_id_pk (primary key), ticket_id_fk (foreign key), transaction_status (status), amount (transaction amount), transaction_date (date), refund_request (refund request status), and user_id_fk (foreign key). Constraints include a foreign key constraint on ticket_id_fk and a primary key constraint on user_id_fk. A comment block at the top provides detailed descriptions for each column. The 'Script Output' panel at the bottom shows the message: 'Table "TRANSACTION_TABLE" already exists. Skipping creation...' and 'PL/SQL procedure successfully completed.'

```

SET SERVEROUTPUT ON;
-- Create TRANSACTION_TABLE if it does not already exist
DECLARE
    table_count INTEGER;
BEGIN
    -- Check if the table exists
    SELECT COUNT(*) INTO table_count FROM all_tables WHERE table_name = 'TRANSACTION_TABLE';

    IF table_count = 0 THEN
        EXECUTE IMMEDIATE '
            CREATE TABLE transaction_table (
                transaction_id_pk INTEGER PRIMARY KEY,
                ticket_id_fk INTEGER,
                transaction_status VARCHAR2(50),
                amount NUMBER(3,2),
                transaction_date DATE,
                refund_request VARCHAR2(50),
                user_id_fk INTEGER
            )
            CONSTRAINT fk_ticket_id FOREIGN KEY (ticket_id_fk) REFERENCES ticket(ticket_id_pk),
            CONSTRAINT fk_user_id FOREIGN KEY (user_id_fk) REFERENCES user_table(user_id_pk)
        ';
        DBMS_OUTPUT.PUT_LINE('Table "TRANSACTION_TABLE" created successfully with refund_request column.');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Table "TRANSACTION_TABLE" already exists. Skipping creation...');
    END IF;
END;
/

```

Step 3 : Execute sequence function only once to generate primary key for each table when insertion happens (use mbta admin)

File path: Functions/ Func_Sequence_PK.sql

The screenshot shows the Oracle SQL Developer interface. The 'Oracle Connections' tree view lists the same set of users as the previous screenshot. The 'Worksheet' tab is active, displaying a PL/SQL script. The code creates a global sequence named 'global_seq_pk' with a start value of 1 and an increment of 1. It includes an exception handling block to skip the creation if the sequence already exists. The 'Script Output' panel at the bottom shows the message: 'Sequence "global_seq" already exists. Skipping...', indicating that the sequence was skipped because it already existed.

```

SET SERVEROUTPUT ON;
-- Create a global sequence to be used by all tables
BEGIN
    EXECUTE IMMEDIATE 'CREATE SEQUENCE global_seq_pk START WITH 1 INCREMENT BY 1';
    DBMS_OUTPUT.PUT_LINE('Sequence "global_seq" created successfully.');
EXCEPTION
    WHEN OTHERS THEN
        IF SQLCODE = -955 THEN
            DBMS_OUTPUT.PUT_LINE('Sequence "global_seq" already exists. Skipping...');
        ELSE
            RAISE;
        END IF;
END;
/

```

Step 4: CREATE Synonym for all the tables using mbta admin

File Path: Roles/Mbta_Table_Synonym_MBTA_Admin.sql

```

Connections Reports | Func_Sequence_PK.sql | Mbta_Table_Synonym_MBTA_Admin.sql
Oracle Connections
  Admin
  cashier_user
  customer_user
  dev
  group_booking_manager_user
  HR
  mbta_admin
  subscription_manager_user
Database Schema Service Connections

Worksheet Query Builder
SET SERVEROUTPUT ON;
-- Create Public Synonyms if they do not exist using mbta admin
DECLARE
  synonym_count INTEGER;
BEGIN
  -- Synonym for user_table
  SELECT COUNT(*) INTO synonym_count FROM all_synonyms WHERE synonym_name = 'USR_TBL' AND owner = 'PUBLIC';
  IF synonym_count = 0 THEN
    EXECUTE IMMEDIATE 'CREATE PUBLIC SYNONYM usr_tbl FOR MBTA_ADMIN.user_table';
    DBMS_OUTPUT.PUT_LINE('Public synonym "USR_TBL" created successfully.');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Public synonym "USR_TBL" already exists. Skipping creation...');

  END IF;

  -- Synonym for discounts
  SELECT COUNT(*) INTO synonym_count FROM all_synonyms WHERE synonym_name = 'DISC' AND owner = 'PUBLIC';
  IF synonym_count = 0 THEN
    EXECUTE IMMEDIATE 'CREATE PUBLIC SYNONYM disc FOR MBTA_ADMIN.discounts';
    DBMS_OUTPUT.PUT_LINE('Public synonym "DISC" created successfully.');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Public synonym "DISC" already exists. Skipping creation...');

  END IF;

  -- Synonym for tknt
  SELECT COUNT(*) INTO synonym_count FROM all_synonyms WHERE synonym_name = 'TKNT' AND owner = 'PUBLIC';
  IF synonym_count = 0 THEN
    EXECUTE IMMEDIATE 'CREATE PUBLIC SYNONYM tknt FOR MBTA_ADMIN.tknt';
    DBMS_OUTPUT.PUT_LINE('Public synonym "TKNT" created successfully.');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Public synonym "TKNT" already exists. Skipping creation...');

  END IF;

  -- Synonym for rds
  SELECT COUNT(*) INTO synonym_count FROM all_synonyms WHERE synonym_name = 'RDS' AND owner = 'PUBLIC';
  IF synonym_count = 0 THEN
    EXECUTE IMMEDIATE 'CREATE PUBLIC SYNONYM rds FOR MBTA_ADMIN.rds';
    DBMS_OUTPUT.PUT_LINE('Public synonym "RDS" created successfully.');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Public synonym "RDS" already exists. Skipping creation...');

  END IF;

  -- Synonym for subbs
  SELECT COUNT(*) INTO synonym_count FROM all_synonyms WHERE synonym_name = 'SUBS' AND owner = 'PUBLIC';
  IF synonym_count = 0 THEN
    EXECUTE IMMEDIATE 'CREATE PUBLIC SYNONYM subbs FOR MBTA_ADMIN.subbs';
    DBMS_OUTPUT.PUT_LINE('Public synonym "SUBS" created successfully.');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Public synonym "SUBS" already exists. Skipping creation...');

  END IF;

  -- Synonym for grp_bkg
  SELECT COUNT(*) INTO synonym_count FROM all_synonyms WHERE synonym_name = 'GRP_BKG' AND owner = 'PUBLIC';
  IF synonym_count = 0 THEN
    EXECUTE IMMEDIATE 'CREATE PUBLIC SYNONYM grp_bkg FOR MBTA_ADMIN.grp_bkg';
    DBMS_OUTPUT.PUT_LINE('Public synonym "GRP_BKG" created successfully.');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Public synonym "GRP_BKG" already exists. Skipping creation...');

  END IF;

END;
  
```

Script Output

```

Task completed in 7.14 seconds
Public synonym "USR_TBL" already exists. Skipping creation...
Public synonym "DISC" already exists. Skipping creation...
Public synonym "TKNT" already exists. Skipping creation...
Public synonym "RDS" already exists. Skipping creation...
Public synonym "SUBS" already exists. Skipping creation...
Public synonym "GRP_BKG" already exists. Skipping creation...

PL/SQL procedure successfully completed.
  
```

STEP 5(DML): Insert Data Into Tables using mbta admin

***** Insert data in the order mentioned**

- File Path: DML/ Insert_Discount_Table.sql:

```

Connections Reports | Insert_Discount_Table.sql
Oracle Connections
  Admin
  cashier_user
  customer_user
  dev
  group_booking_manager_user
  HR
  mbta_admin
  subscription_manager_user
Database Schema Service Connections

Worksheet Query Builder
BEGIN
  -- Insert discount with discount_id_pk = 0 (Regular)
  SELECT COUNT(*) INTO discount_exists FROM discounts WHERE discount_id_pk = 0;
  IF discount_exists = 0 THEN
    INSERT INTO discounts (discount_id_pk, discount_type, discount_rate)
    VALUES (0, 'Regular', 0);
    DBMS_OUTPUT.PUT_LINE('Discount "Regular" with ID 0 inserted.');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Discount "Regular" with ID 0 already exists.');
  END IF;

  -- Insert discount with discount_id_pk = 1 (Student)
  SELECT COUNT(*) INTO discount_exists FROM discounts WHERE discount_id_pk = 1;
  IF discount_exists = 0 THEN
    INSERT INTO discounts (discount_id_pk, discount_type, discount_rate)
    VALUES (1, 'Student', 5.00);
    DBMS_OUTPUT.PUT_LINE('Discount "Student" with ID 1 inserted.');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Discount "Student" with ID 1 already exists.');
  END IF;

  -- Insert discount with discount_id_pk = 2 (Senior)
  SELECT COUNT(*) INTO discount_exists FROM discounts WHERE discount_id_pk = 2;
  IF discount_exists = 0 THEN
    INSERT INTO discounts (discount_id_pk, discount_type, discount_rate)
    VALUES (2, 'Senior', 10.00);
    DBMS_OUTPUT.PUT_LINE('Discount "Senior" with ID 2 inserted.');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Discount "Senior" with ID 2 already exists.');
  END IF;
END;
  
```

Script Output

```

Task completed in 0.196 seconds
Discount "Regular" with ID 0 already exists.
Discount "Student" with ID 1 already exists.
Discount "Senior" with ID 2 already exists.
Discount "Group" with ID 3 already exists.

PL/SQL procedure successfully completed.
  
```

- File Path: DML/ Insert_User_Table.sql

- File Path: DML/ Func_Sequence_PK.sql

```

SELECT COUNT(*) INTO user_exists FROM usr_tbl WHERE email_uniq = 'user8@example.com';
IF user_exists = 0 THEN
  INSERT INTO usr_tbl (user_id_pk, username, password, email_uniq, user_category, created_at, discount_id_fk)
  VALUES (user_seq_pk.NEXTVAL, 'user_8', 'password_8', 'user8@example.com', 'Student', SYSDATE - 8, 1);
  DBMS_OUTPUT.PUT_LINE('User 8 inserted.');
ELSE
  DBMS_OUTPUT.PUT_LINE('User 8 already exists. Skipping insertion...');

-- Insert User 9
SELECT COUNT(*) INTO user_exists FROM usr_tbl WHERE email_uniq = 'user9@example.com';
IF user_exists = 0 THEN
  INSERT INTO usr_tbl (user_id_pk, username, password, email_uniq, user_category, created_at, discount_id_fk)
  VALUES (user_seq_pk.NEXTVAL, 'user_9', 'password_9', 'user9@example.com', 'Regular', SYSDATE - 9, 0);
  DBMS_OUTPUT.PUT_LINE('User 9 inserted.');
ELSE
  DBMS_OUTPUT.PUT_LINE('User 9 already exists. Skipping insertion...');

-- Insert User 10
SELECT COUNT(*) INTO user_exists FROM usr_tbl WHERE email_uniq = 'user10@example.com';
IF user_exists = 0 THEN
  INSERT INTO usr_tbl (user_id_pk, username, password, email_uniq, user_category, created_at, discount_id_fk)
  VALUES (user_seq_pk.NEXTVAL, 'user_10', 'password_10', 'user10@example.com', 'Senior', SYSDATE - 10, 2);
  DBMS_OUTPUT.PUT_LINE('User 10 inserted.');
ELSE
  DBMS_OUTPUT.PUT_LINE('User 10 already exists. Skipping insertion...');
END IF;
END;
  
```

Script Output x | Task completed in 0.21 seconds

User 3 already exists. Skipping insertion...
User 4 already exists. Skipping insertion...
User 5 already exists. Skipping insertion...
User 6 already exists. Skipping insertion...
User 7 already exists. Skipping insertion...
User 8 already exists. Skipping insertion...
User 9 already exists. Skipping insertion...
User 10 already exists. Skipping insertion...

PL/SQL procedure successfully completed.

- File Path: DML/ Insert_Ticket_Table.sql

```

SET SERVEROUTPUT ON;

-- Insert data into TICKET table using global_seq_pk sequence for primary key
DECLARE
  ticket_exists INTEGER;
BEGIN
  -- Insert Ticket 1
  SELECT COUNT(*) INTO ticket_exists FROM ticket WHERE user_id_fk = 1 AND purchase_date = SYSDATE - 1;
  IF ticket_exists = 0 THEN
    INSERT INTO ticket (ticket_id_pk, user_id_fk, purchase_date, price, transit_line, ticket_status)
    VALUES (ticket_seq_pk.NEXTVAL, 1, SYSDATE - 1, 2.40, 'Red Line', 'Valid');
    DBMS_OUTPUT.PUT_LINE('Ticket 1 inserted.');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Ticket 1 already exists. Skipping insertion...');

  -- Insert Ticket 2
  SELECT COUNT(*) INTO ticket_exists FROM ticket WHERE user_id_fk = 2 AND purchase_date = SYSDATE - 2;
  IF ticket_exists = 0 THEN
    INSERT INTO ticket (ticket_id_pk, user_id_fk, purchase_date, price, transit_line, ticket_status)
    VALUES (ticket_seq_pk.NEXTVAL, 2, SYSDATE - 2, 2.40, 'Green Line', 'Cancelled');
    DBMS_OUTPUT.PUT_LINE('Ticket 2 inserted.');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Ticket 2 already exists. Skipping insertion...');

  -- Insert Ticket 3
  
```

Script Output x | Task completed in 0.355 seconds

Ticket 1 inserted.
Ticket 2 inserted.
Ticket 3 inserted.
Ticket 4 inserted.
Ticket 5 inserted.

PL/SQL procedure successfully completed.

- File Path: DML/ Insert_Transcation_Table.sql

File Path: DML/ Insert_Transcation_Table.sql

```

SQL Worksheet History
Worksheet - Query Builder
  DECLARE
    transaction_exists INTEGER;
  BEGIN
    Insert Transaction 1: Completed with Not Eligible refund
    SELECT COUNT(*) INTO transaction_exists FROM transaction_table WHERE ticket_id_fk = 1 AND transaction_date = SYSDATE - 1;
    IF transaction_exists = 0 THEN
      INSERT INTO transaction_table (transaction_id_pk, ticket_id_fk, transaction_status, amount, transaction_date, refund_status, refund_request, user_id_fk)
      VALUES (transaction_seq_pk.NEXTVAL, 1, 'Completed', 2.40, SYSDATE - 1, 'Not Refunded', 'Not Requested', 1);
      DBMS_OUTPUT.PUT_LINE('Transaction 1 inserted: Completed with Not Eligible refund.');
    ELSE
      DBMS_OUTPUT.PUT_LINE('Transaction 1 already exists. Skipping insertion...');

    END IF;

    -- Insert Transaction 2: Pending, refund request is not eligible by default
    SELECT COUNT(*) INTO transaction_exists FROM transaction_table WHERE ticket_id_fk = 2 AND transaction_date = SYSDATE - 2;
    IF transaction_exists = 0 THEN
      INSERT INTO transaction_table (transaction_id_pk, ticket_id_fk, transaction_status, amount, transaction_date, refund_status, refund_request, user_id_fk)
      VALUES (transaction_seq_pk.NEXTVAL, 2, 'Pending', 2.40, SYSDATE - 2, 'Not Eligible', 'Not Requested', 2);
      DBMS_OUTPUT.PUT_LINE('Transaction 2 inserted: Pending, Not Eligible for refund.');
    ELSE
      DBMS_OUTPUT.PUT_LINE('Transaction 2 already exists. Skipping insertion...');

    END IF;

    -- Insert Transaction 3: Completed with Refunded status and refund request
    SELECT COUNT(*) INTO transaction_exists FROM transaction_table WHERE ticket_id_fk = 3 AND transaction_date = SYSDATE - 3;
    IF transaction_exists = 0 THEN
      ...
    END IF;
  END;

```

Script Output

Task completed in 0.624 seconds

Transaction 1 inserted: Completed, Not Eligible refund.
 Transaction 2 inserted: Pending, Not Eligible for refund.
 Transaction 3 inserted: Completed with refund processed.
 Transaction 4 inserted: Cancelled with auto refund request.
 Transaction 5 inserted: Completed with Eligible refund.

PL/SQL procedure successfully completed.

- File Path: DML/ Insert_Group_Bookings_Table.sql

```

SET SERVEROUTPUT ON;
DECLARE
  booking_exists INTEGER;
  user_exists INTEGER;
  discount_exists INTEGER;
BEGIN
  Ensure that discount ID 3 exists in the DISCOUNTS table before proceeding
  SELECT COUNT(*) INTO discount_exists FROM discounts WHERE discount_id_pk = 3;

  IF discount_exists = 0 THEN
    DBMS_OUTPUT.PUT_LINE('Discount ID 3 does not exist. Cannot proceed with group booking inserts.');
    RETURN;
  END IF;

  -- Insert Group Booking 1
  SELECT COUNT(*) INTO user_exists FROM user_table WHERE user_id_pk = 1;
  SELECT COUNT(*) INTO booking_exists FROM group_booking WHERE user_id_fk = 1 AND booking_date = SYSDATE - 1;

  IF user_exists > 0 AND booking_exists = 0 THEN
    INSERT INTO group_booking (group_booking_id_pk, user_id_fk, num_passengers, booking_date, transit_line, transaction_status, discounts_discount_id_pk)
    VALUES (groupBooking_seq_pk.NEXTVAL, 1, 5, SYSDATE - 1, 'Red Line', 'Approved', 3);
    DBMS_OUTPUT.PUT_LINE('Group Booking 1 inserted.');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Group Booking 1 already exists or has invalid references. Skipping insertion...');

  END IF;

  -- Insert Group Booking 2
  SELECT COUNT(*) INTO user_exists FROM user_table WHERE user_id_pk = 2;
  SELECT COUNT(*) INTO booking_exists FROM group_booking WHERE user_id_fk = 2 AND booking_date = SYSDATE - 1;

  IF user_exists > 0 AND booking_exists = 0 THEN
    INSERT INTO group_booking (group_booking_id_pk, user_id_fk, num_passengers, booking_date, transit_line, transaction_status, discounts_discount_id_pk)
    VALUES (groupBooking_seq_pk.NEXTVAL, 2, 5, SYSDATE - 1, 'Blue Line', 'Approved', 3);
    DBMS_OUTPUT.PUT_LINE('Group Booking 2 inserted.');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Group Booking 2 already exists or has invalid references. Skipping insertion...');

  END IF;

  -- Insert Group Booking 3
  SELECT COUNT(*) INTO user_exists FROM user_table WHERE user_id_pk = 3;
  SELECT COUNT(*) INTO booking_exists FROM group_booking WHERE user_id_fk = 3 AND booking_date = SYSDATE - 1;

  IF user_exists > 0 AND booking_exists = 0 THEN
    INSERT INTO group_booking (group_booking_id_pk, user_id_fk, num_passengers, booking_date, transit_line, transaction_status, discounts_discount_id_pk)
    VALUES (groupBooking_seq_pk.NEXTVAL, 3, 5, SYSDATE - 1, 'Green Line', 'Approved', 3);
    DBMS_OUTPUT.PUT_LINE('Group Booking 3 inserted.');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Group Booking 3 already exists or has invalid references. Skipping insertion...');

  END IF;

  -- Insert Group Booking 4
  SELECT COUNT(*) INTO user_exists FROM user_table WHERE user_id_pk = 4;
  SELECT COUNT(*) INTO booking_exists FROM group_booking WHERE user_id_fk = 4 AND booking_date = SYSDATE - 1;

  IF user_exists > 0 AND booking_exists = 0 THEN
    INSERT INTO group_booking (group_booking_id_pk, user_id_fk, num_passengers, booking_date, transit_line, transaction_status, discounts_discount_id_pk)
    VALUES (groupBooking_seq_pk.NEXTVAL, 4, 5, SYSDATE - 1, 'Yellow Line', 'Approved', 3);
    DBMS_OUTPUT.PUT_LINE('Group Booking 4 inserted.');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Group Booking 4 already exists or has invalid references. Skipping insertion...');

  END IF;

  -- Insert Group Booking 5
  SELECT COUNT(*) INTO user_exists FROM user_table WHERE user_id_pk = 5;
  SELECT COUNT(*) INTO booking_exists FROM group_booking WHERE user_id_fk = 5 AND booking_date = SYSDATE - 1;

  IF user_exists > 0 AND booking_exists = 0 THEN
    INSERT INTO group_booking (group_booking_id_pk, user_id_fk, num_passengers, booking_date, transit_line, transaction_status, discounts_discount_id_pk)
    VALUES (groupBooking_seq_pk.NEXTVAL, 5, 5, SYSDATE - 1, 'Purple Line', 'Approved', 3);
    DBMS_OUTPUT.PUT_LINE('Group Booking 5 inserted.');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Group Booking 5 already exists or has invalid references. Skipping insertion...');

  END IF;

```

Script Output

Task completed in 0.177 seconds

Group Booking 1 inserted.
 Group Booking 2 inserted.
 Group Booking 3 inserted.
 Group Booking 4 inserted.
 Group Booking 5 inserted.

PL/SQL procedure successfully completed.

- File Path: DML/ Insert_Rides_Table.sql

```

SET SERVEROUTPUT ON;
DECLARE
  ride_exists INTEGER;
BEGIN
  -- Insert Ride 1
  SELECT COUNT(*) INTO ride_exists FROM rides WHERE user_id_fk = 1 AND ride_timestamp = SYSDATE - 1;
  IF ride_exists = 0 THEN
    INSERT INTO rides (ride_id, user_id_fk, ride_timestamp, total_spend)
    VALUES (rides_seq_pk.NEXTVAL, 1, SYSDATE - 1, 20);
    DBMS_OUTPUT.PUT_LINE('Ride 1 inserted.');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Ride 1 already exists. Skipping insertion...');

  END IF;

  -- Insert Ride 2
  SELECT COUNT(*) INTO ride_exists FROM rides WHERE user_id_fk = 2 AND ride_timestamp = SYSDATE - 2;
  IF ride_exists = 0 THEN
    INSERT INTO rides (ride_id, user_id_fk, ride_timestamp, total_spend)
    VALUES (rides_seq_pk.NEXTVAL, 2, SYSDATE - 2, 30);
    DBMS_OUTPUT.PUT_LINE('Ride 2 inserted.');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Ride 2 already exists. Skipping insertion...');

  END IF;

  -- Insert Ride 3
  SELECT COUNT(*) INTO ride_exists FROM rides WHERE user_id_fk = 3 AND ride_timestamp = SYSDATE - 3;
  IF ride_exists = 0 THEN
    ...
  END IF;

```

Script Output

Task completed in 0.331 seconds

Ride 1 inserted.
 Ride 2 inserted.
 Ride 3 inserted.
 Ride 4 inserted.
 Ride 5 inserted.

PL/SQL procedure successfully completed.

File Path: DML/ Insert_Subscription_Table.sql

```

SQL Worksheet | History
[Run] [Stop] [Save] [New] [Open] [Close] [Help] | 0.23999999 seconds

Worksheet | Query Builder
SET SERVEROUTPUT ON;
DECLARE
    subscription_exists INTEGER;
BEGIN
    -- Insert Subscription for User 1
    SELECT COUNT(*) INTO subscription_exists FROM subscription WHERE user_id_fk = 5 AND subscription_type = 'Monthly';
    IF subscription_exists = 0 THEN
        INSERT INTO subscription (subscription_id_pk, user_id_fk, subscription_type, start_date, end_date, subscription_status)
        VALUES (subscription_seq_pk.NEXTVAL, 5, 'Monthly', SYSDATE - 30, SYSDATE, 'Active');
        DBMS_OUTPUT.PUT_LINE('Subscription for User 1 inserted.');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Subscription for User 1 already exists. Skipping insertion...');
    END IF;

    -- Insert Subscription for User 2
    SELECT COUNT(*) INTO subscription_exists FROM subscription WHERE user_id_fk = 6 AND subscription_type = 'Yearly';
    IF subscription_exists = 0 THEN
        INSERT INTO subscription (subscription_id_pk, user_id_fk, subscription_type, start_date, end_date, subscription_status)
        VALUES (subscription_seq_pk.NEXTVAL, 6, 'Yearly', SYSDATE - 365, SYSDATE + 365, 'Active');
        DBMS_OUTPUT.PUT_LINE('Subscription for User 2 inserted.');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Subscription for User 2 already exists. Skipping insertion...');
    END IF;

    -- Insert Subscription for User 3
    SELECT COUNT(*) INTO subscription_exists FROM subscription WHERE user_id_fk = 7 AND subscription_type = 'Monthly';
    IF subscription_exists = 0 THEN
        ...
    END IF;
END;

```

Script Output x

Task completed in 0.24 seconds

```

Subscription for User 1 inserted.
Subscription for User 2 inserted.
Subscription for User 3 inserted.
Subscription for User 4 inserted.
Subscription for User 5 inserted.

PL/SQL procedure successfully completed.

```

Step 6: Create Users and Roles (login as mbta admin)

File Path: Role/ User_Creation_and_Grant_Roles.sql

```

Oracle Connections
  Admin
  cashier_user
  customer_user
  dev
  group_booking_manager_user
  HR
  mbta_admin
  subscription_manager_user
Database Schema Service Connections

Worksheet | Query Builder
-- Enable server output to display messages
SET SERVEROUTPUT ON;

-- Create Roles
DECLARE
    role_name VARCHAR2(30);
BEGIN
    -- Customer Role
    role_name := 'CUSTOMER_ROLE';
    BEGIN
        EXECUTE IMMEDIATE 'CREATE ROLE ' || role_name;
        DBMS_OUTPUT.PUT_LINE('Role ' || role_name || ' created successfully.');
    EXCEPTION
        WHEN OTHERS THEN
            IF SQLCODE = -1921 THEN
                DBMS_OUTPUT.PUT_LINE('Role ' || role_name || ' already exists. Skipping...');

            ELSE
                RAISE;
            END IF;
    END;

    -- Cashier Role
    role_name := 'CASHIER_ROLE';
    BEGIN
        EXECUTE IMMEDIATE 'CREATE ROLE ' || role_name;
        DBMS_OUTPUT.PUT_LINE('Role ' || role_name || ' created successfully.');
    EXCEPTION
        WHEN OTHERS THEN
            ...
    END;

    -- GROUP_BOOKING_MANAGER_ROLE
    role_name := 'GROUP_BOOKING_MANAGER_ROLE';
    BEGIN
        EXECUTE IMMEDIATE 'CREATE ROLE ' || role_name;
        DBMS_OUTPUT.PUT_LINE('Role ' || role_name || ' created successfully.');
    EXCEPTION
        WHEN OTHERS THEN
            ...
    END;

    -- SUBSCRIPTION_MANAGER_ROLE
    role_name := 'SUBSCRIPTION_MANAGER_ROLE';
    BEGIN
        EXECUTE IMMEDIATE 'CREATE ROLE ' || role_name;
        DBMS_OUTPUT.PUT_LINE('Role ' || role_name || ' created successfully.');
    EXCEPTION
        WHEN OTHERS THEN
            ...
    END;
END;

```

Script Output x

Task completed in 0.381 seconds

```

Role CUSTOMER_ROLE already exists. Skipping...
Role CASHIER_ROLE already exists. Skipping...
Role GROUP_BOOKING_MANAGER_ROLE already exists. Skipping...
Role SUBSCRIPTION_MANAGER_ROLE already exists. Skipping...

PL/SQL procedure successfully completed.

Granted SELECT on ticket and subscription, INSERT on ticket to CUSTOMER_ROLE.
Granted SELECT and UPDATE on transaction_table to CASHIER_ROLE.
Granted SELECT and UPDATE on group_booking to GROUP_BOOKING_MANAGER_ROLE.
Granted SELECT on subscription to SUBSCRIPTION_MANAGER_ROLE.

PL/SQL procedure successfully completed.

User "customer_user" already exists. Skipping password update...
User "cashier_user" already exists. Skipping password update...
User "group_booking_manager_user" already exists. Skipping password update...
User "subscription_manager_user" already exists. Skipping password update...

PL/SQL procedure successfully completed.

```

Step 7: Create Views (Login as Mbta admin) (exec all views as admin)

File Path : Views/Views_Creation_mbta_admin/

The screenshot shows the Oracle SQL Developer interface with the 'Ticket_Sales_By_Line_Category_View.sql' script open in the worksheet. The code creates a view named 'TICKET_SALES_BY_LINE_CATEGORY' by selecting the count of tickets sold per user category and transit line. It includes logic to check if the view exists and drop it if it does. The script ends with a DBMS_OUTPUT message confirming the view was created successfully.

```

-- Create Ticket_Sales_By_Line_Category View
DECLARE
    v_count NUMBER;
BEGIN
    -- Check if the view exists
    SELECT COUNT(*) INTO v_count FROM user_views WHERE view_name = 'TICKET_SALES_BY_LINE_CATEGORY';

    -- Drop the view if it exists
    IF v_count > 0 THEN
        EXECUTE IMMEDIATE 'DROP VIEW TICKET_SALES_BY_LINE_CATEGORY';
    END IF;

    -- Recreate the view
    EXECUTE IMMEDIATE '
CREATE VIEW TICKET_SALES_BY_LINE_CATEGORY AS
SELECT
    t.transit_line,
    u.user_category,
    COUNT(t.ticket_id_pk) AS ticket_count
FROM
    MBTA_ADMIN.ticket t
    JOIN MBTA_ADMIN.user_table u ON t.user_id_fk = u.user_id_pk
GROUP BY
    t.transit_line, u.user_category';
    DBMS_OUTPUT.PUT_LINE('Ticket_Sales_By_Line_Category view created successfully.');
END;
/

```

Script Output:

Task completed in 0.22 seconds
Ticket_Sales_By_Line_Category view created successfully.
PL/SQL procedure successfully completed.

Step 8: Execute VIEWS related permissions (LOGIN AS MBTA ADMIN)

File Path: Role/ Views_permission_mbta_admin.sql

The screenshot shows the Oracle SQL Developer interface with the 'Views_permission_mbta_admin.sql' script open in the worksheet. The script contains multiple GRANT SELECT statements for various views across different roles: ADMIN_ROLE, CASHIER_ROLE, and CUSTOMER_ROLE. It grants SELECT permissions on views like 'Subscription_Utilization', 'Transaction_Summary', 'Ticket_Sales_By_Line_Category', 'Refunds_Cancellation_Summary', and 'Customer_Ticket_Subscription'. The script ends with a DBMS_OUTPUT message confirming all grants were successful.

```

-- Grant SELECT on Subscription_Utilization to ADMIN_ROLE and SUBSCRIPTION_MANAGER_ROLE
EXECUTE IMMEDIATE 'GRANT SELECT ON MBTA_ADMIN.Subscription_Utilization TO ADMIN_ROLE';
DBMS_OUTPUT.PUT_LINE('Granted SELECT on Subscription_Utilization to ADMIN_ROLE');
EXECUTE IMMEDIATE 'GRANT SELECT ON MBTA_ADMIN.Subscription_Utilization TO SUBSCRIPTION_MANAGER_ROLE';
DBMS_OUTPUT.PUT_LINE('Granted SELECT on Subscription_Utilization to SUBSCRIPTION_MANAGER_ROLE');

-- Grant SELECT on Transaction_Summary to ADMIN_ROLE and CASHIER_ROLE
EXECUTE IMMEDIATE 'GRANT SELECT ON MBTA_ADMIN.Transaction_Summary TO ADMIN_ROLE';
DBMS_OUTPUT.PUT_LINE('Granted SELECT on Transaction_Summary to ADMIN_ROLE');
EXECUTE IMMEDIATE 'GRANT SELECT ON MBTA_ADMIN.Transaction_Summary TO CASHIER_ROLE';
DBMS_OUTPUT.PUT_LINE('Granted SELECT on Transaction_Summary to CASHIER_ROLE');

-- Grant SELECT on Ticket_Sales_By_Line_Category to ADMIN_ROLE
EXECUTE IMMEDIATE 'GRANT SELECT ON MBTA_ADMIN.Ticket_Sales_By_Line_Category TO ADMIN_ROLE';
DBMS_OUTPUT.PUT_LINE('Granted SELECT on Ticket_Sales_By_Line_Category to ADMIN_ROLE');

-- Grant SELECT on Refund_Cancellation_Summary to CASHIER_ROLE and ADMIN_ROLE
EXECUTE IMMEDIATE 'GRANT SELECT ON MBTA_ADMIN.Refunds_Cancellation_Summary TO CASHIER_ROLE';
DBMS_OUTPUT.PUT_LINE('Granted SELECT on Refunds_Cancellation_Summary to CASHIER_ROLE');
EXECUTE IMMEDIATE 'GRANT SELECT ON MBTA_ADMIN.Refunds_Cancellation_Summary TO ADMIN_ROLE';
DBMS_OUTPUT.PUT_LINE('Granted SELECT on Refunds_Cancellation_Summary to ADMIN_ROLE');

-- Grant SELECT on Customer_Ticket_Subscription to CUSTOMER_ROLE
EXECUTE IMMEDIATE 'GRANT SELECT ON MBTA_ADMIN.Customer_Ticket_Subscription TO CUSTOMER_ROLE';
DBMS_OUTPUT.PUT_LINE('Granted SELECT on Customer_Ticket_Subscription to CUSTOMER_ROLE');

END;
/

```

Script Output:

Task completed in 0.267 seconds
Granted SELECT on Subscription_Utilization to ADMIN_ROLE.
Granted SELECT on Subscription_Utilization to SUBSCRIPTION_MANAGER_ROLE.
Granted SELECT on Transaction_Summary to ADMIN_ROLE.
Granted SELECT on Transaction_Summary to CASHIER_ROLE.
Granted SELECT on Ticket_Sales_By_Line_Category to ADMIN_ROLE.
Granted SELECT on Refunds_Cancellation_Summary to CASHIER_ROLE.
Granted SELECT on Refunds_Cancellation_Summary to ADMIN_ROLE.
Granted SELECT on Customer_Ticket_Subscription to CUSTOMER_ROLE.
PL/SQL procedure successfully completed.
Granted SELECT on Customer_Summary to ADMIN_ROLE.
Granted SELECT on Group_Booking_Overview to GROUP_BOOKING_MANAGER_ROLE.
Granted SELECT on Subscription_Utilization to ADMIN_ROLE.
Granted SELECT on Subscription_Utilization to SUBSCRIPTION_MANAGER_ROLE.
Granted SELECT on Transaction_Summary to ADMIN_ROLE.
Granted SELECT on Transaction_Summary to CASHIER_ROLE.
Granted SELECT on Ticket_Sales_By_Line_Category to ADMIN_ROLE.
Granted SELECT on Refunds_Cancellation_Summary to CASHIER_ROLE.
Granted SELECT on Refunds_Cancellation_Summary to ADMIN_ROLE.
Granted SELECT on Customer_Ticket_Subscription to CUSTOMER_ROLE.
PL/SQL procedure successfully completed.

Step 9: Exec Views

File Path : Views/Exec_VIEWS/ Exec_group_booking_view_gbManager
Login as group booking manager

The screenshot shows an Oracle SQL Worksheet interface. The title bar has three tabs: 'Views_permission_mbta_admin.sql', 'Exec_Customer_Summary_mbta_admin.sql', and 'Exec_group_booking_view_gbManager.sql'. The main area is titled 'Worksheet' and contains the following PL/SQL code:

```
-- Group_Booking_Overview View
DECLARE
    CURSOR cur_group_booking IS
        SELECT * FROM MBTA_ADMIN.Group_Booking_Overview;
    rec_group_booking cur_group_booking%ROWTYPE;
BEGIN
    DBMS_OUTPUT.PUT_LINE('--- Group_Booking_Overview View ---');
    OPEN cur_group_booking;
    LOOP
        FETCH cur_group_booking INTO rec_group_booking;
        EXIT WHEN cur_group_booking%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('Booking ID: ' || rec_group_booking.booking_id ||
                            ', Number of Passengers: ' || rec_group_booking.num_passenger ||
                            ', Booking Date: ' || rec_group_booking.booking_date ||
                            ', Transit Line: ' || rec_group_booking.transit_line ||
                            ', Booking Status: ' || rec_group_booking.booking_status);
    END LOOP;
    CLOSE cur_group_booking;
END;
/
```

The 'Script Output' pane at the bottom shows the results of the query:

```
--- Group_Booking_Overview View ---
Booking ID: 1, Number of Passengers: 5, Booking Date: 07-NOV-24, Transit Line: Red Line, Booking Status: Approved
Booking ID: 2, Number of Passengers: 10, Booking Date: 06-NOV-24, Transit Line: Green Line, Booking Status: Denied
Booking ID: 3, Number of Passengers: 7, Booking Date: 05-NOV-24, Transit Line: Blue Line, Booking Status: Approved
Booking ID: 4, Number of Passengers: 15, Booking Date: 04-NOV-24, Transit Line: Orange Line, Booking Status: Pending
Booking ID: 5, Number of Passengers: 14, Booking Date: 29-OCT-24, Transit Line: Green Line, Booking Status: Approved
Booking ID: 6, Number of Passengers: 5, Booking Date: 07-NOV-24, Transit Line: Red Line, Booking Status: Approved
Booking ID: 7, Number of Passengers: 10, Booking Date: 06-NOV-24, Transit Line: Green Line, Booking Status: Denied
Booking ID: 8, Number of Passengers: 7, Booking Date: 05-NOV-24, Transit Line: Blue Line, Booking Status: Approved
Booking ID: 9, Number of Passengers: 15, Booking Date: 04-NOV-24, Transit Line: Orange Line, Booking Status: Pending
Booking ID: 10, Number of Passengers: 14, Booking Date: 29-OCT-24, Transit Line: Green Line, Booking Status: Approved
```

A message at the bottom indicates the procedure completed successfully.

File Path: Views/Exec_VIEWS/
Exec_Subscription_utilization_view_Subscription_manager.sql

...ql Exec_group_booking_view_gbManager.sql × Exec_Subscription_utilization_view_Subscription_manager.sql × Exec_Trans

SQL Worksheet History 0.38499999 seconds

Worksheet Query Builder

```
-- exec as Subscription manager to view Subscription_Utilization
SET SERVEROUTPUT ON;

-- Subscription_Utilization View
DECLARE
  CURSOR cur_subscription_utilization IS
    SELECT * FROM MBTA_ADMIN.Subscription_Utilization;
  rec_subscription cur_subscription_utilization%ROWTYPE;
BEGIN
  DBMS_OUTPUT.PUT_LINE('--- Subscription_Utilization View ---');
  OPEN cur_subscription_utilization;
  LOOP
    FETCH cur_subscription_utilization INTO rec_subscription;
    EXIT WHEN cur_subscription_utilization%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('Subscription ID: ' || rec_subscription.subscription_id ||
      ', User ID: ' || rec_subscription.user_id ||
      ', Subscription Type: ' || rec_subscription.subscription_type ||
      ', Start Date: ' || rec_subscription.start_date ||
      ', End Date: ' || rec_subscription.end_date ||
      ', Subscription Status: ' || rec_subscription.subscription_status);
  END LOOP;
  CLOSE cur_subscription_utilization;
END;
/

```

Script Output Task completed in 0.385 seconds

```
--- Subscription_Utilization View ---
Subscription ID: 1, User ID: 5, Subscription Type: Monthly, Start Date: 09-OCT-24, End Date: 08-NOV-24, Subscription Status: Active
Subscription ID: 2, User ID: 6, Subscription Type: Yearly, Start Date: 09-NOV-23, End Date: 08-NOV-25, Subscription Status: Active
Subscription ID: 3, User ID: 7, Subscription Type: Monthly, Start Date: 24-OCT-24, End Date: 23-NOV-24, Subscription Status: Expired
Subscription ID: 4, User ID: 7, Subscription Type: Weekly, Start Date: 01-NOV-24, End Date: 08-NOV-24, Subscription Status: Active
Subscription ID: 5, User ID: 8, Subscription Type: Monthly, Start Date: 24-SEP-24, End Date: 24-OCT-24, Subscription Status: Expired

PL/SQL procedure successfully completed.
```

File Path: Views/Exec_VIEWS/ Exec_Transaction_Summary_view_cashier_user.sql

...ql Exec_group_booking_view_gbManager.sql × Exec_Subscription_utilization_view_Subscription_manager.sql × Exec_Transaction_Summary_view_cashier_user.sql

SQL Worksheet History 0.31999999 seconds

Worksheet Query Builder

```
-- exec as cashier user to view Transaction_Summary & Refunds_Cancellation_Summary
SET SERVEROUTPUT ON;

-- Transaction_Summary View
DECLARE
  CURSOR cur_transaction_summary IS
    SELECT * FROM MBTA_ADMIN.Transaction_Summary;
  rec_transaction_summary cur_transaction_summary%ROWTYPE;
BEGIN
  DBMS_OUTPUT.PUT_LINE('--- Transaction_Summary View ---');
  OPEN cur_transaction_summary;
  LOOP
    FETCH cur_transaction_summary INTO rec_transaction_summary;
    EXIT WHEN cur_transaction_summary%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('Transaction ID: ' || rec_transaction_summary.transaction_id ||
      ', Ticket ID: ' || rec_transaction_summary.ticket_id ||
      ', Transaction Status: ' || rec_transaction_summary.transaction_status ||
      ', Amount: ' || rec_transaction_summary.amount ||
      ', Transaction Date: ' || rec_transaction_summary.transaction_date ||
      ', Refund Status: ' || rec_transaction_summary.refund_status ||
      ', Refund Request: ' || rec_transaction_summary.refund_request ||
      ', User ID: ' || rec_transaction_summary.user_id);
  END LOOP;
  CLOSE cur_transaction_summary;
END;
/

```

Script Output Task completed in 0.32 seconds

```
--- Transaction_Summary View ---
Transaction ID: 1, Ticket ID: 1, Transaction Status: Completed, Amount: 2.4, Transaction Date: 07-NOV-24, Refund Status: Not Refunded, Refund Request: Not Requested, User ID: 1
Transaction ID: 2, Ticket ID: 2, Transaction Status: Pending, Amount: 2.4, Transaction Date: 06-NOV-24, Refund Status: Not Eligible, Refund Request: Not Requested, User ID: 2
Transaction ID: 3, Ticket ID: 3, Transaction Status: Completed, Amount: 2.4, Transaction Date: 05-NOV-24, Refund Status: Refunded, Refund Request: Requested, User ID: 3
Transaction ID: 4, Ticket ID: 4, Transaction Status: Cancelled, Amount: 2.4, Transaction Date: 04-NOV-24, Refund Status: Eligible, Refund Request: Requested, User ID: 4
Transaction ID: 5, Ticket ID: 5, Transaction Status: Completed, Amount: 2.4, Transaction Date: 03-NOV-24, Refund Status: Eligible, Refund Request: Requested, User ID: 5

PL/SQL procedure successfully completed.

--- Refunds_Cancellation_Summary View ---
Transaction ID: 3, Ticket ID: 3, Refunded Amount: 2.4, Refund Status: Refunded, Transaction Date: 05-NOV-24
Transaction ID: 4, Ticket ID: 4, Refunded Amount: 2.4, Refund Status: Eligible, Transaction Date: 04-NOV-24

PL/SQL procedure successfully completed.
```