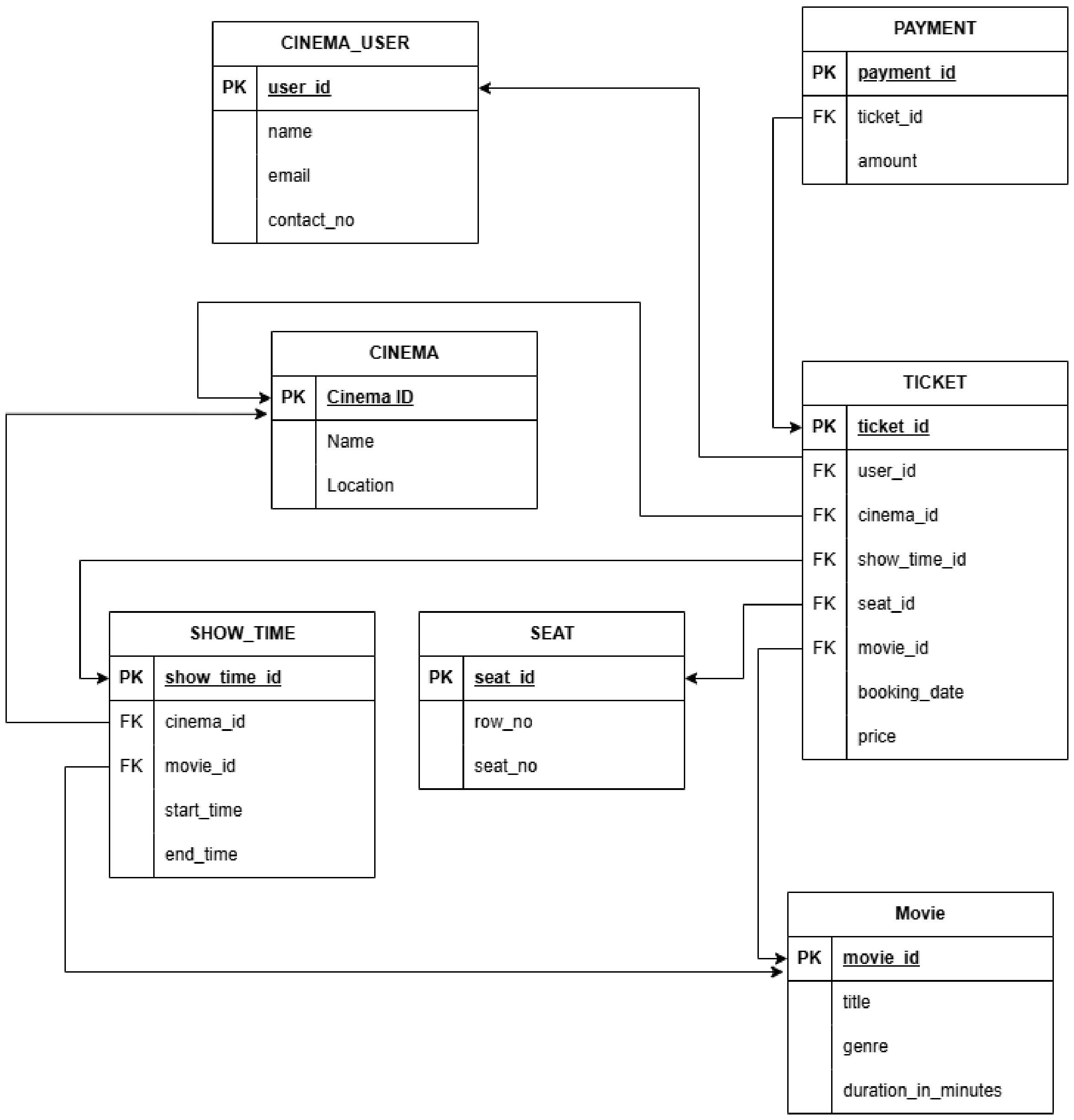
Cinema Booking System Implementation with Oracle PL/SQL

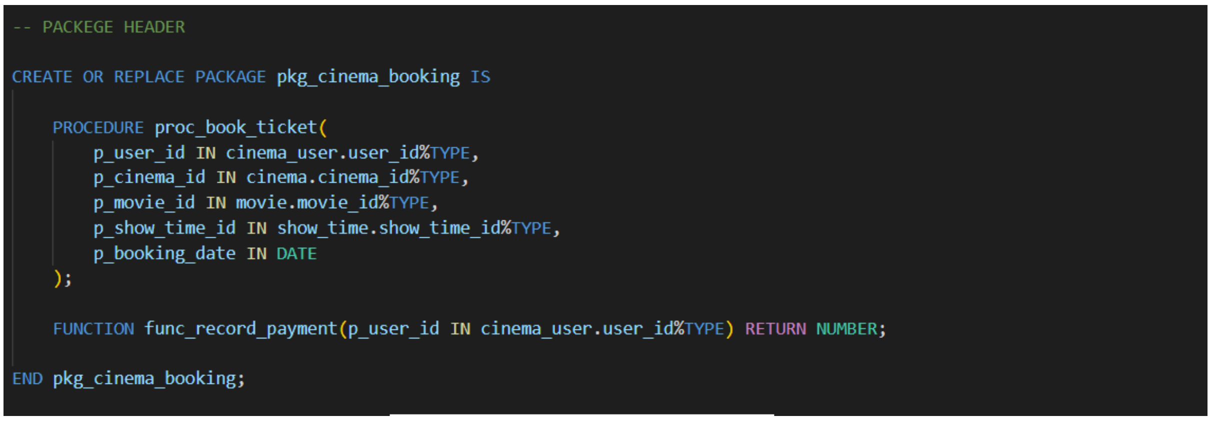
Our Cinema Booking System is designed as a ticket booking system for cinemas. By this System, User can book a ticket to watch a movie from a cinema that user want. Also this system provides several facilities other than handling ticket bookings. It also provides seat allocations without conflicts, handle payments properly, and keeps logs for booking activity and also prevent access to book tickets for invalidate users allowing registers users to book seats for movies to relevant show times.

This System is also an efficient platform for users to browse available movies, select comfortable show times, and get to know cinemas and the relevant locations and prices of the movies that the user wants to enjoy. Our cinema booking system is also a platform to seek movies under the genre as the user wants and the duration of the relevant movie. This system helps to minimize the manual effort, human errors and reduce paper waste while improving user satisfaction, efficiency and accuracy.



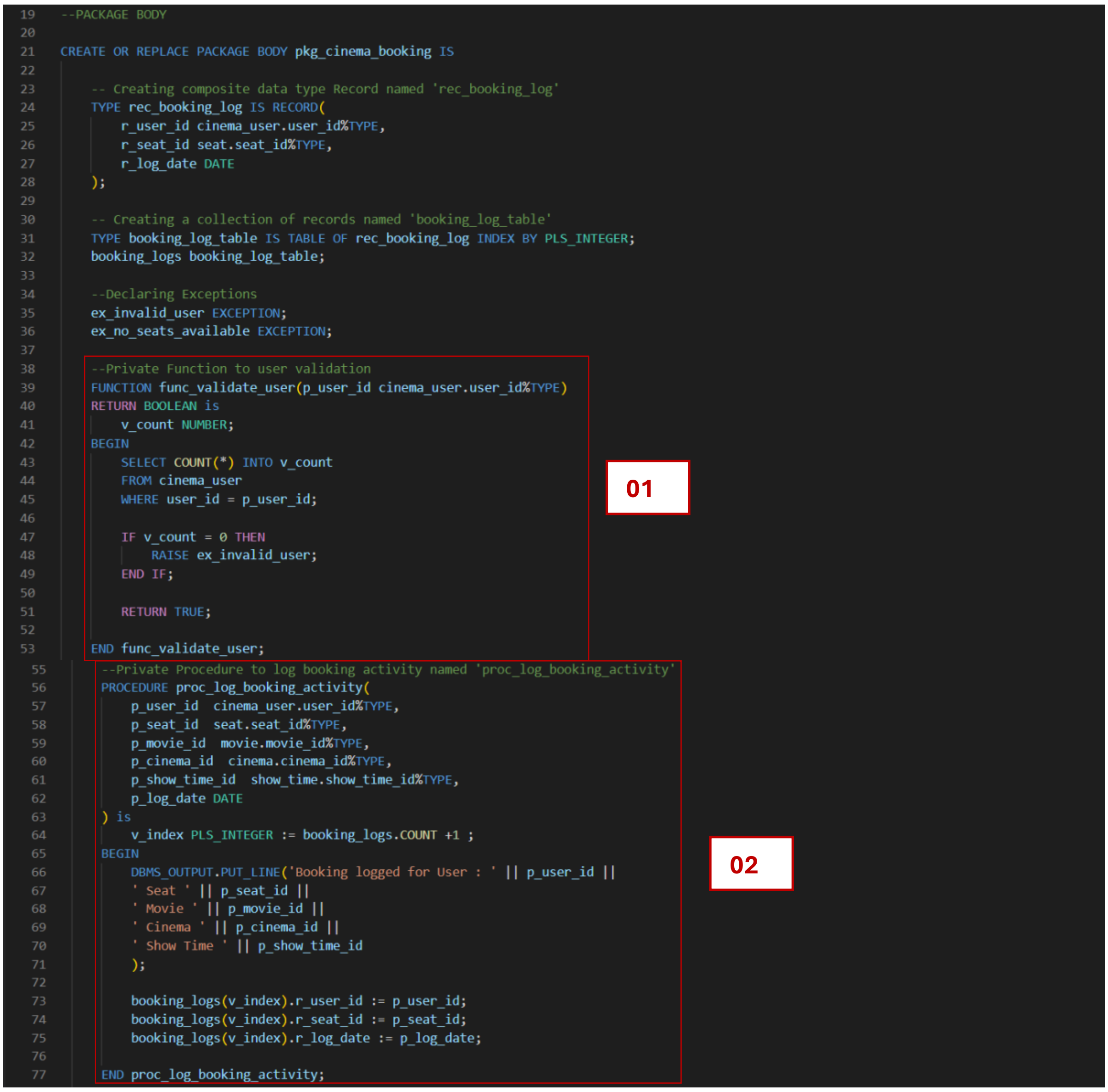
Relational Model Diagram

Package *pkg_cinema_booking* Components

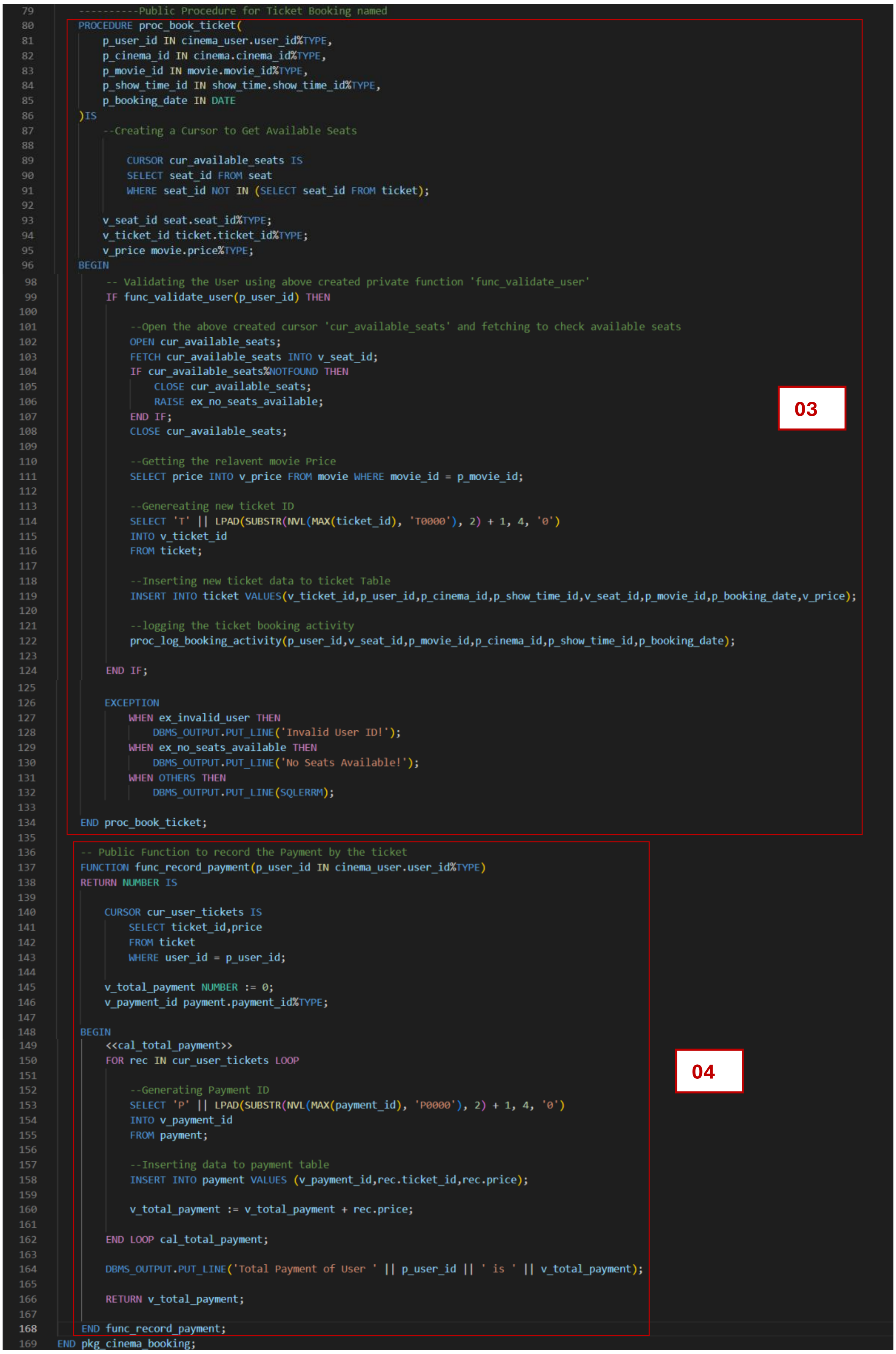


```sql
-- PACKEGE HEADER

CREATE OR REPLACE PACKAGE pkg_cinema_booking IS

    PROCEDURE proc_book_ticket(
        p_user_id IN cinema_user.user_id%TYPE,
        p_cinema_id IN cinema.cinema_id%TYPE,
        p_movie_id IN movie.movie_id%TYPE,
        p_show_time_id IN show_time.show_time_id%TYPE,
        p_booking_date IN DATE
    );

    FUNCTION func_record_payment(p_user_id IN cinema_user.user_id%TYPE) RETURN NUMBER;

END pkg_cinema_booking;
```

Package Header



```sql
--PACKAGE BODY

CREATE OR REPLACE PACKAGE BODY pkg_cinema_booking IS

    -- Creating composite data type Record named 'rec_booking_log'
    TYPE rec_booking_log IS RECORD(
        r_user_id cinema_user.user_id%TYPE,
        r_seat_id seat.seat_id%TYPE,
        r_log_date DATE
    );

    -- Creating a collection of records named 'booking_log_table'
    TYPE booking_log_table IS TABLE OF rec_booking_log INDEX BY PLS_INTEGER;
    booking_logs booking_log_table;

    --Declaring Exceptions
    ex_invalid_user EXCEPTION;
    ex_no_seats_available EXCEPTION;

    --Private Function to user validation
    FUNCTION func_validate_user(p_user_id cinema_user.user_id%TYPE)
    RETURN BOOLEAN is
        v_count NUMBER;
    BEGIN
        SELECT COUNT(*) INTO v_count
        FROM cinema_user
        WHERE user_id = p_user_id;

        IF v_count = 0 THEN
            RAISE ex_invalid_user;
        END IF;

        RETURN TRUE;

    END func_validate_user;
    --Private Procedure to log booking activity named 'proc_log_booking_activity'
    PROCEDURE proc_log_booking_activity(
        p_user_id  cinema_user.user_id%TYPE,
        p_seat_id  seat.seat_id%TYPE,
        p_movie_id  movie.movie_id%TYPE,
        p_cinema_id  cinema.cinema_id%TYPE,
        p_show_time_id  show_time.show_time_id%TYPE,
        p_log_date DATE
    ) is
        v_index PLS_INTEGER := booking_logs.COUNT +1 ;
    BEGIN
        DBMS_OUTPUT.PUT_LINE('Booking logged for User : ' || p_user_id ||
        ' Seat ' || p_seat_id ||
        ' Movie ' || p_movie_id ||
        ' Cinema ' || p_cinema_id ||
        ' Show Time ' || p_show_time_id
        );

        booking_logs(v_index).r_user_id := p_user_id;
        booking_logs(v_index).r_seat_id := p_seat_id;
        booking_logs(v_index).r_log_date := p_log_date;

    END proc_log_booking_activity;
```



```sql
    ----------Public Procedure for Ticket Booking named
    PROCEDURE proc_book_ticket(
        p_user_id IN cinema_user.user_id%TYPE,
        p_cinema_id IN cinema.cinema_id%TYPE,
        p_movie_id IN movie.movie_id%TYPE,
        p_show_time_id IN show_time.show_time_id%TYPE,
        p_booking_date IN DATE
    )IS
        --Creating a Cursor to Get Available Seats

            CURSOR cur_available_seats IS
            SELECT seat_id FROM seat
            WHERE seat_id NOT IN (SELECT seat_id FROM ticket);

        v_seat_id seat.seat_id%TYPE;
        v_ticket_id ticket.ticket_id%TYPE;
        v_price movie.price%TYPE;
    BEGIN
        -- Validating the User using above created private function 'func_validate_user'
        IF func_validate_user(p_user_id) THEN

            --Open the above created cursor 'cur_available_seats' and fetching to check available seats
            OPEN cur_available_seats;
            FETCH cur_available_seats INTO v_seat_id;
            IF cur_available_seats%NOTFOUND THEN
                CLOSE cur_available_seats;
                RAISE ex_no_seats_available;
            END IF;
            CLOSE cur_available_seats;

            --Getting the relavent movie Price
            SELECT price INTO v_price FROM movie WHERE movie_id = p_movie_id;

            --Genereating new ticket ID
            SELECT 'T' || LPAD(SUBSTR(NVL(MAX(ticket_id), 'T0000'), 2) + 1, 4, '0')
            INTO v_ticket_id
            FROM ticket;

            --Inserting new ticket data to ticket Table
            INSERT INTO ticket VALUES(v_ticket_id,p_user_id,p_cinema_id,p_show_time_id,v_seat_id,p_movie_id,p_booking_date,v_price);

            --logging the ticket booking activity
            proc_log_booking_activity(p_user_id,v_seat_id,p_movie_id,p_cinema_id,p_show_time_id,p_booking_date);

        END IF;

        EXCEPTION
            WHEN ex_invalid_user THEN
                DBMS_OUTPUT.PUT_LINE('Invalid User ID!');
            WHEN ex_no_seats_available THEN
                DBMS_OUTPUT.PUT_LINE('No Seats Available!');
            WHEN OTHERS THEN
                DBMS_OUTPUT.PUT_LINE(SQLERRM);

    END proc_book_ticket;

    -- Public Function to record the Payment by the ticket
    FUNCTION func_record_payment(p_user_id IN cinema_user.user_id%TYPE)
    RETURN NUMBER IS

        CURSOR cur_user_tickets IS
            SELECT ticket_id,price
            FROM ticket
            WHERE user_id = p_user_id;

        v_total_payment NUMBER := 0;
        v_payment_id payment.payment_id%TYPE;

    BEGIN
        <<cal_total_payment>>
        FOR rec IN cur_user_tickets LOOP

            --Generating Payment ID
            SELECT 'P' || LPAD(SUBSTR(NVL(MAX(payment_id), 'P0000'), 2) + 1, 4, '0')
            INTO v_payment_id
            FROM payment;

            --Inserting data to payment table
            INSERT INTO payment VALUES (v_payment_id,rec.ticket_id,rec.price);

            v_total_payment := v_total_payment + rec.price;

        END LOOP cal_total_payment;

        DBMS_OUTPUT.PUT_LINE('Total Payment of User ' || p_user_id || ' is ' || v_total_payment);

        RETURN v_total_payment;

    END func_record_payment;
END pkg_cinema_booking;
```

Package Body

This package consists of a private function, private procedure, public procedure and a public function.

1. Private Function - *func_validate_user*

This function is used to validate a user that means prevent bookings for unregistered users. It uses a SELECT query to check whether the relevant user id is available at CINEMA_USER table and gets the count. Then the function uses IF control statement to check whether the count is 0.

If the count is 0, it means user is not in the CINEMA_USER table and the relevant user is not a registered user. Then it raises an exception that, provided user id is a *'invalid user'*. Otherwise, the function returns TRUE.

2. Private Procedure - *proc_log_booking_activity*

This procedure is used to log the booking activity of the relevant user. It uses a collection of records to store the log details of booking. And it outputs a statement about the logging details, Such as user id, seat id, movie id , cinema id and show time id.

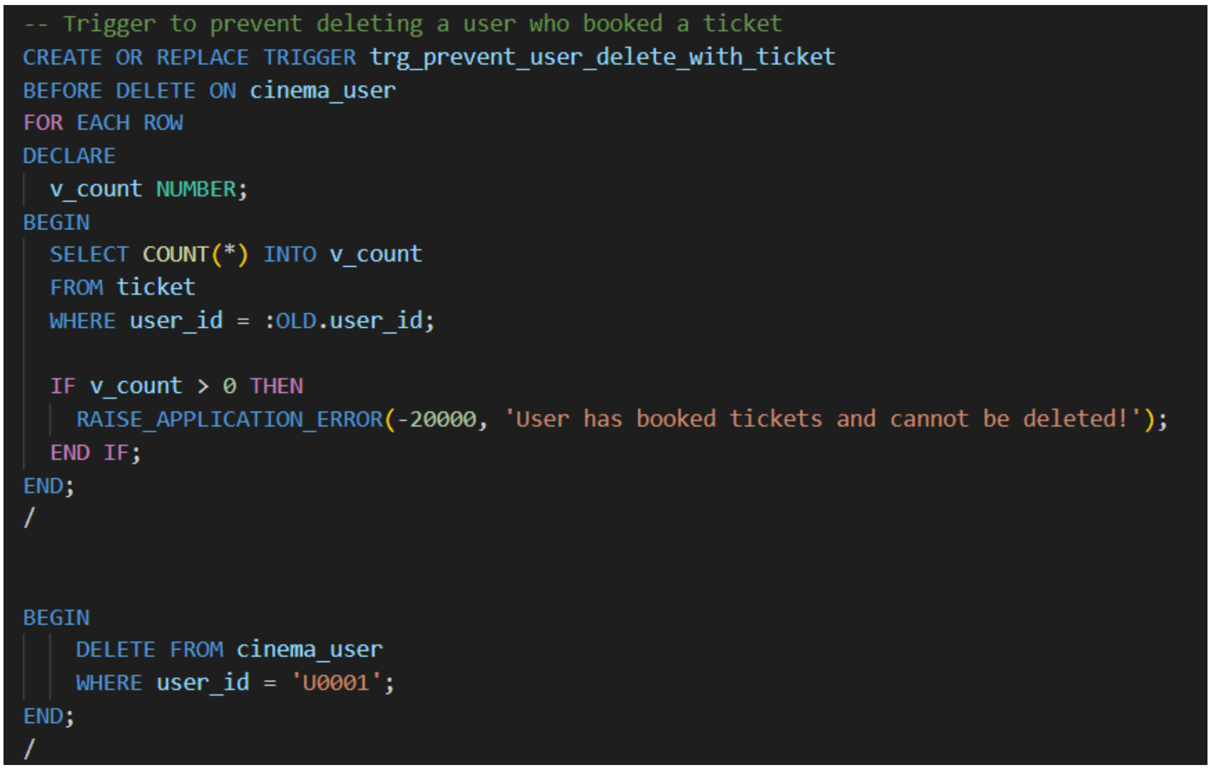
3. Public Procedure - *proc_book_ticket*

The function of this procedure is it book a ticket for the user. By this procedure it reserves a seat from the available seats for the relevant user and issues a ticket id for that user and that data is added to the ticket table. When consider about the reservation of a seat for a user, It uses a cursor called *cur_available_seats* to get the available seats from the seat table, It checks the seat ids that are already booked and included in the ticket table. Also, this procedure calls for the function *func_validate_user* to check the user is valid, if it valid then uses the cursor to check available seats, if the seats are not available, it raises and exception, showing the error message 'No Seats Available!'. Then it fetches the price for the movie that user want from the movie table, and generate a new ticket ID. Finally, it enters the new data to the ticket table and calls the procedure *proc_log_booking_activity* for keep the log for the new booking and display the booking details.

4. Public Function - *func_record_payment*

This function is used to calculate the total payment for all the tickets booked by a user and inserts the ticket data and the payment to the payment table. This function declares a cursor called *cur_user_tickets* to get the ticket id and the price for a relevant user. Then it uses a cursor for loop to loop through the cursor, and, inside the loop, it generates a new payment ID and insert the new data into the payment table using the new payment ID. Finally, it also displays the total payment for the relevant user.

Trigger : *trg_prevent_user_delete_with_ticket*

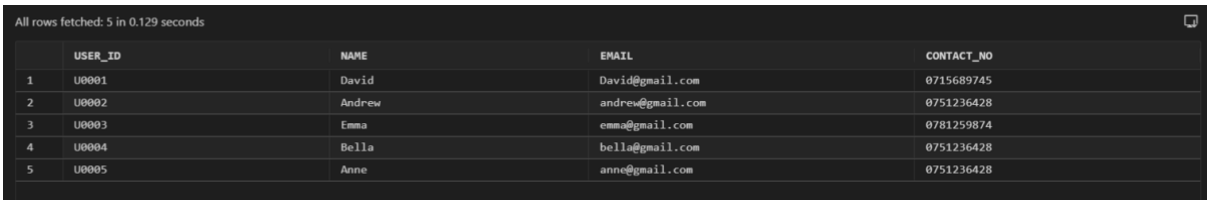


```
-- Trigger to prevent deleting a user who booked a ticket
CREATE OR REPLACE TRIGGER trg_prevent_user_delete_with_ticket
BEFORE DELETE ON cinema_user
FOR EACH ROW
DECLARE
  v_count NUMBER;
BEGIN
  SELECT COUNT(*) INTO v_count
  FROM ticket
  WHERE user_id = :OLD.user_id;

  IF v_count > 0 THEN
    RAISE_APPLICATION_ERROR(-20000, 'User has booked tickets and cannot be deleted!');
  END IF;
END;
/


BEGIN
    DELETE FROM cinema_user
    WHERE user_id = 'U0001';
END;
/
```

The function of this trigger is it prevents the deletion of a user who has already booked tickets. By this trigger, before a record is deleted from the CINEMA_USER table, it checks whether that user has any tickets recorded in the ticket table. It executes a SELECT COUNT(*) query to count the number of tickets the user has booked, using the user ID from the row that is about to be deleted. If the count is greater than 0, it raises an error using RAISE_APPLICATION_ERROR with a custom error message ‘User has booked tickets and cannot be deleted!’.

Test Outputs

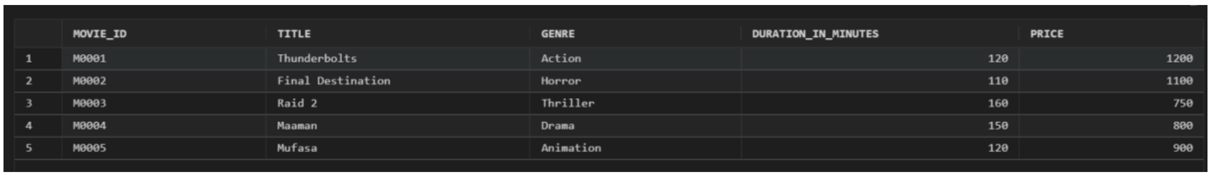
CINEMA_USER Table



All rows fetched: 5 in 0.129 seconds

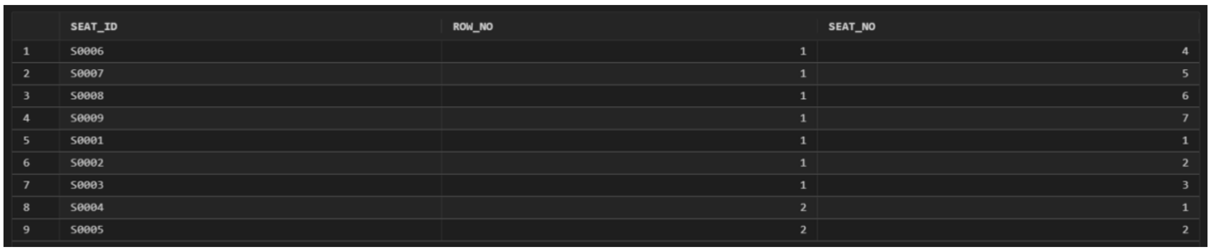
	USER_ID	NAME	EMAIL	CONTACT_NO
1	U0001	David	David@gmail.com	0715689745
2	U0002	Andrew	andrew@gmail.com	0751236428
3	U0003	Emma	emma@gmail.com	0781259874
4	U0004	Bella	bella@gmail.com	0751236428
5	U0005	Anne	anne@gmail.com	0751236428

MOVIE Table



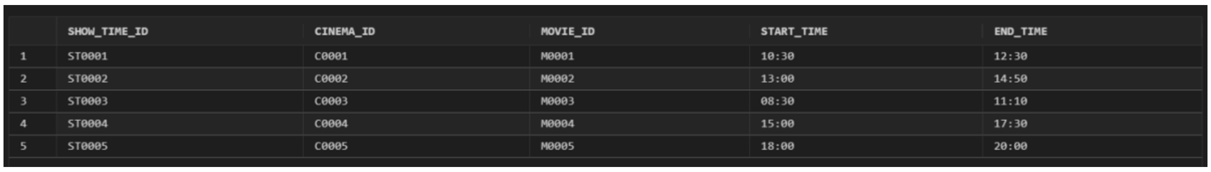
	MOVIE_ID	TITLE	GENRE	DURATION_IN_MINUTES	PRICE
1	M0001	Thunderbolts	Action	120	1200
2	M0002	Final Destination	Horror	110	1100
3	M0003	Raid 2	Thriller	160	750
4	M0004	Maaman	Drama	150	800
5	M0005	Mufasa	Animation	120	900

SEAT Table



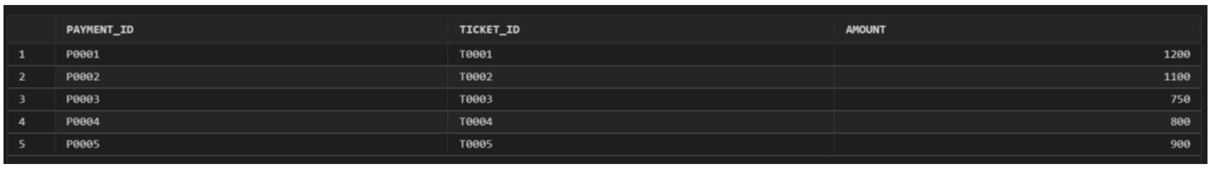
	SEAT_ID	ROW_NO	SEAT_NO
1	S0006	1	4
2	S0007	1	5
3	S0008	1	6
4	S0009	1	7
5	S0001	1	1
6	S0002	1	2
7	S0003	1	3
8	S0004	2	1
9	S0005	2	2

SHOW_TIME Table



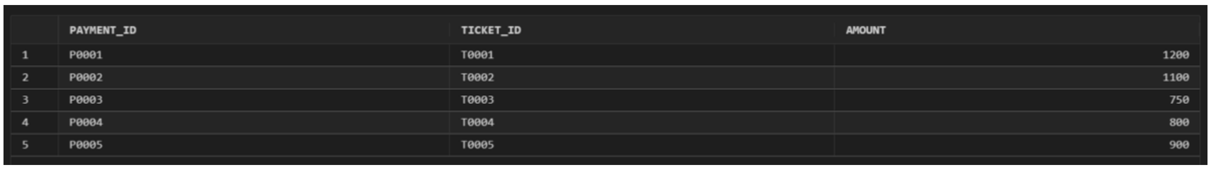
	SHOW_TIME_ID	CINEMA_ID	MOVIE_ID	START_TIME	END_TIME
1	ST0001	C0001	M0001	10:30	12:30
2	ST0002	C0002	M0002	13:00	14:50
3	ST0003	C0003	M0003	08:30	11:10
4	ST0004	C0004	M0004	15:00	17:30
5	ST0005	C0005	M0005	18:00	20:00

PAYMENT Table



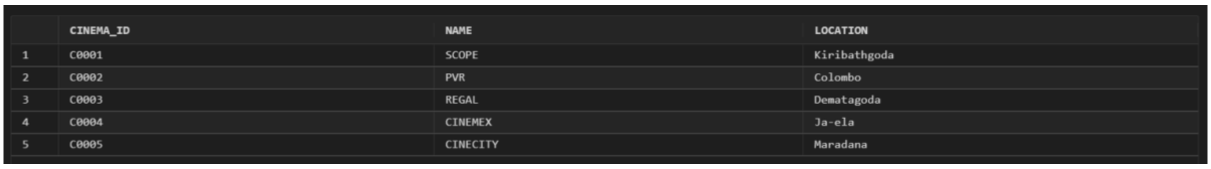
	PAYMENT_ID	TICKET_ID	AMOUNT
1	P0001	T0001	1200
2	P0002	T0002	1100
3	P0003	T0003	750
4	P0004	T0004	800
5	P0005	T0005	900

TICKET Table



	PAYMENT_ID	TICKET_ID	AMOUNT
1	P0001	T0001	1200
2	P0002	T0002	1100
3	P0003	T0003	750
4	P0004	T0004	800
5	P0005	T0005	900

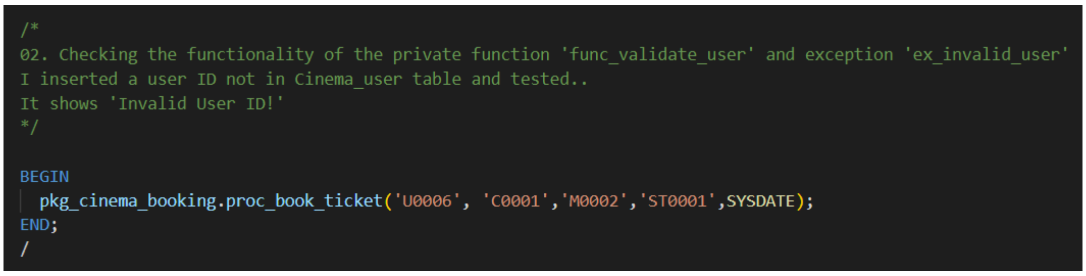
CINEMA Table



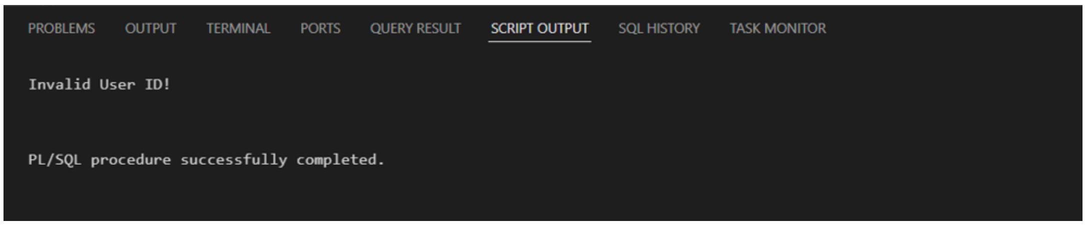
	CINEMA_ID	NAME	LOCATION
1	C0001	SCOPE	Kiribathgoda
2	C0002	PVR	Colombo
3	C0003	REGAL	Dematagoda
4	C0004	CINEMEX	Ja-ela
5	C0005	CINECITY	Maradana

Testing the Procedure Components

Test function : *func_validate_user*

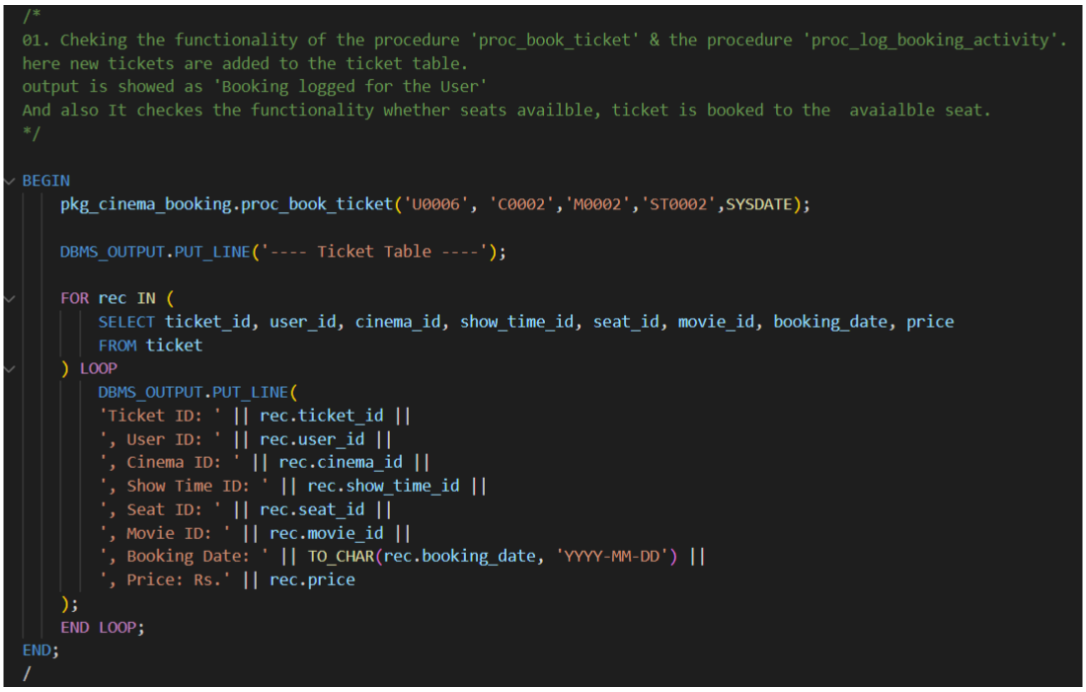


```
/*
02. Checking the functionality of the private function 'func_validate_user' and exception 'ex_invalid_user'
I inserted a user ID not in Cinema_user table and tested..
It shows 'Invalid User ID!'
*/

BEGIN
  pkg_cinema_booking.proc_book_ticket('U0006', 'C0001','M0002','ST0001',SYSDATE);
END;
/
```

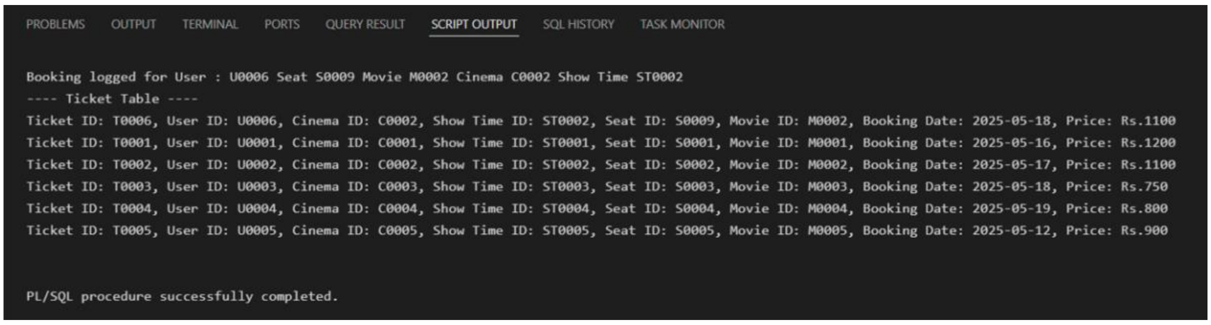


```
Invalid User ID!


PL/SQL procedure successfully completed.
```

Test Procedure : *proc_log_booking_activity* / *proc_book_ticket*

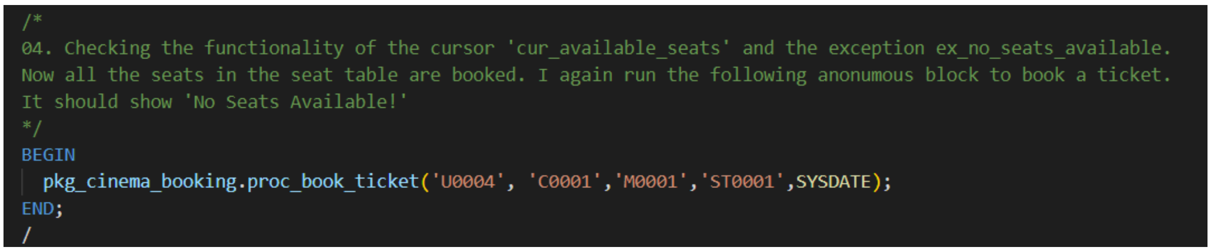


```
/*
01. Cheking the functionality of the procedure 'proc_book_ticket' & the procedure 'proc_log_booking_activity'.
here new tickets are added to the ticket table.
output is showed as 'Booking logged for the User'
And also It checkes the functionality whether seats availble, ticket is booked to the  avaialble seat.
*/

BEGIN
    pkg_cinema_booking.proc_book_ticket('U0006', 'C0002','M0002','ST0002',SYSDATE);

    DBMS_OUTPUT.PUT_LINE('---- Ticket Table ----');

    FOR rec IN (
        SELECT ticket_id, user_id, cinema_id, show_time_id, seat_id, movie_id, booking_date, price
        FROM ticket
    ) LOOP
        DBMS_OUTPUT.PUT_LINE(
        'Ticket ID: ' || rec.ticket_id ||
        ', User ID: ' || rec.user_id ||
        ', Cinema ID: ' || rec.cinema_id ||
        ', Show Time ID: ' || rec.show_time_id ||
        ', Seat ID: ' || rec.seat_id ||
        ', Movie ID: ' || rec.movie_id ||
        ', Booking Date: ' || TO_CHAR(rec.booking_date, 'YYYY-MM-DD') ||
        ', Price: Rs.' || rec.price
    );
    END LOOP;
END;
/
```

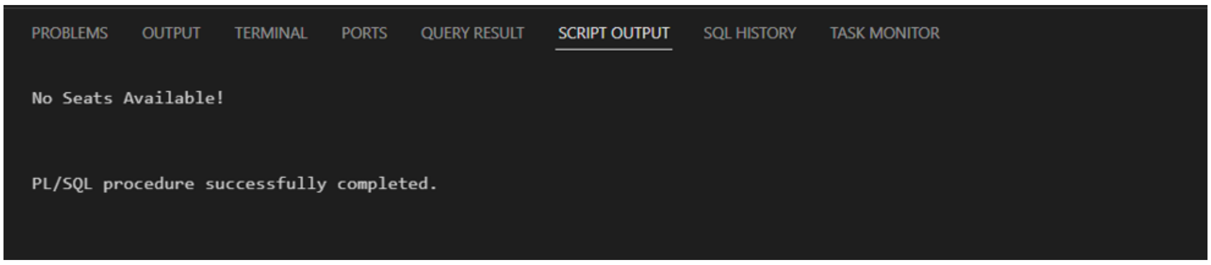


```
Booking logged for User : U0006 Seat S0009 Movie M0002 Cinema C0002 Show Time ST0002
---- Ticket Table ----
Ticket ID: T0006, User ID: U0006, Cinema ID: C0002, Show Time ID: ST0002, Seat ID: S0009, Movie ID: M0002, Booking Date: 2025-05-18, Price: Rs.1100
Ticket ID: T0001, User ID: U0001, Cinema ID: C0001, Show Time ID: ST0001, Seat ID: S0001, Movie ID: M0001, Booking Date: 2025-05-16, Price: Rs.1200
Ticket ID: T0002, User ID: U0002, Cinema ID: C0002, Show Time ID: ST0002, Seat ID: S0002, Movie ID: M0002, Booking Date: 2025-05-17, Price: Rs.1100
Ticket ID: T0003, User ID: U0003, Cinema ID: C0003, Show Time ID: ST0003, Seat ID: S0003, Movie ID: M0003, Booking Date: 2025-05-18, Price: Rs.750
Ticket ID: T0004, User ID: U0004, Cinema ID: C0004, Show Time ID: ST0004, Seat ID: S0004, Movie ID: M0004, Booking Date: 2025-05-19, Price: Rs.800
Ticket ID: T0005, User ID: U0005, Cinema ID: C0005, Show Time ID: ST0005, Seat ID: S0005, Movie ID: M0005, Booking Date: 2025-05-12, Price: Rs.900


PL/SQL procedure successfully completed.
```

Test Cursor : *cur_available_seats*

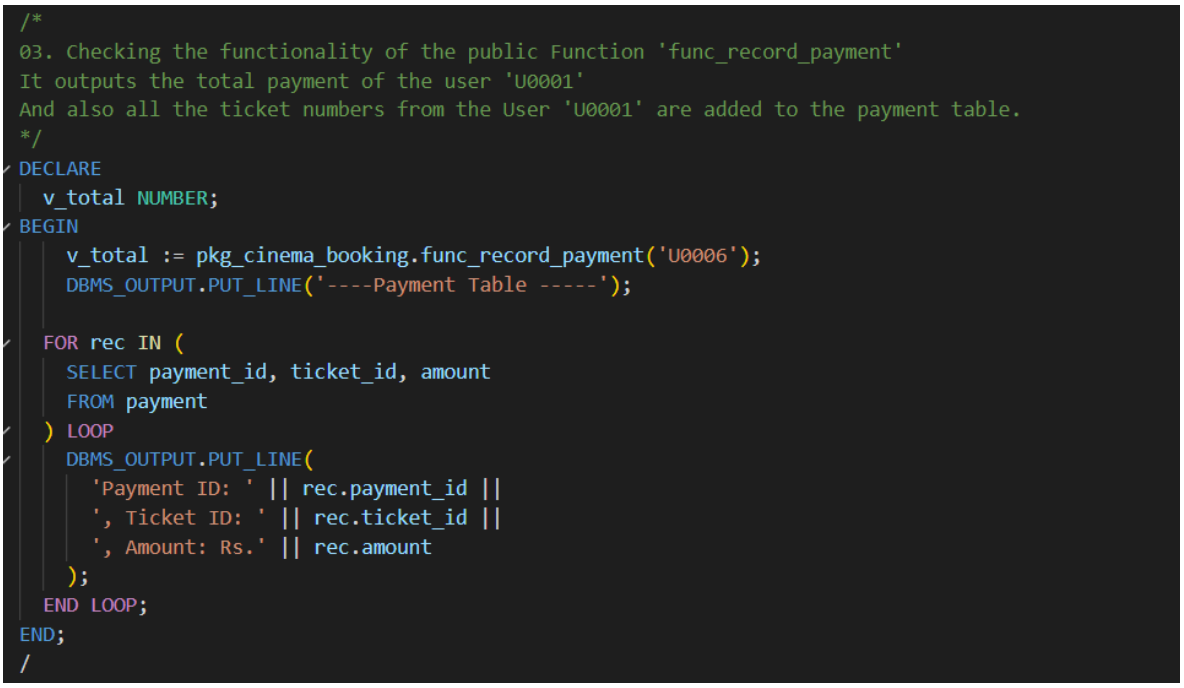


```
/*
04. Checking the functionality of the cursor 'cur_available_seats' and the exception ex_no_seats_available.
Now all the seats in the seat table are booked. I again run the following anonumous block to book a ticket.
It should show 'No Seats Available!'
*/
BEGIN
  pkg_cinema_booking.proc_book_ticket('U0004', 'C0001','M0001','ST0001',SYSDATE);
END;
/
```

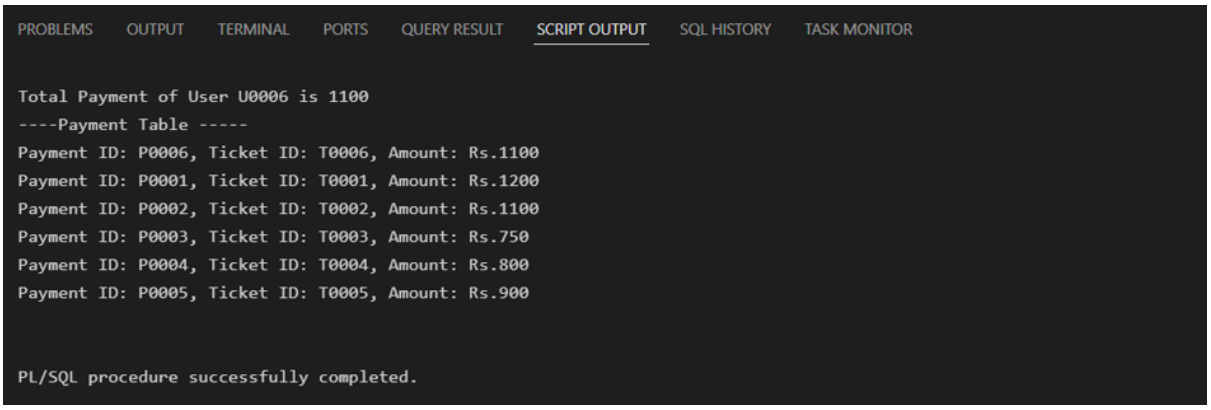


```
No Seats Available!


PL/SQL procedure successfully completed.
```

Test Function : *func_record_payment*

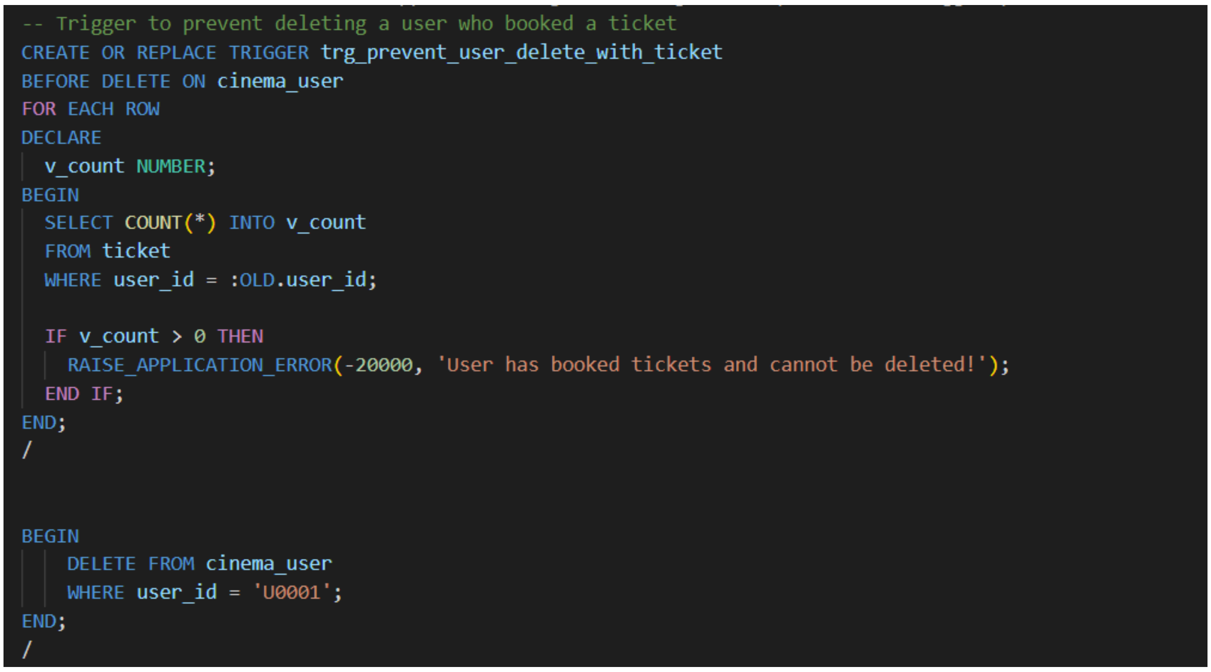


```
/*
03. Checking the functionality of the public Function 'func_record_payment'
It outputs the total payment of the user 'U0001'
And also all the ticket numbers from the User 'U0001' are added to the payment table.
*/
DECLARE
  v_total NUMBER;
BEGIN
    v_total := pkg_cinema_booking.func_record_payment('U0006');
    DBMS_OUTPUT.PUT_LINE('----Payment Table -----');

  FOR rec IN (
    SELECT payment_id, ticket_id, amount
    FROM payment
  ) LOOP
    DBMS_OUTPUT.PUT_LINE(
      'Payment ID: ' || rec.payment_id ||
      ', Ticket ID: ' || rec.ticket_id ||
      ', Amount: Rs.' || rec.amount
    );
  END LOOP;
END;
/
```

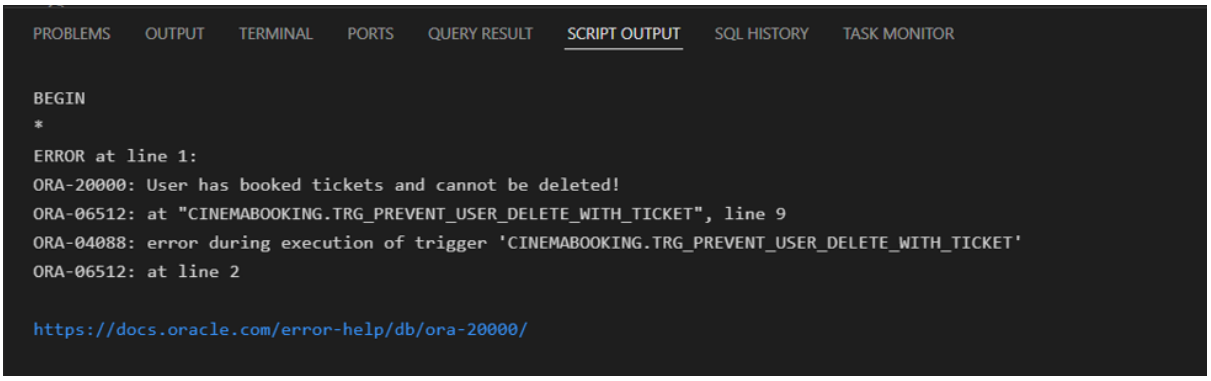


```
Total Payment of User U0006 is 1100
----Payment Table -----
Payment ID: P0006, Ticket ID: T0006, Amount: Rs.1100
Payment ID: P0001, Ticket ID: T0001, Amount: Rs.1200
Payment ID: P0002, Ticket ID: T0002, Amount: Rs.1100
Payment ID: P0003, Ticket ID: T0003, Amount: Rs.750
Payment ID: P0004, Ticket ID: T0004, Amount: Rs.800
Payment ID: P0005, Ticket ID: T0005, Amount: Rs.900


PL/SQL procedure successfully completed.
```

Testing the Trigger : *trg_prevent_user_delete_with_ticket*



```
-- Trigger to prevent deleting a user who booked a ticket
CREATE OR REPLACE TRIGGER trg_prevent_user_delete_with_ticket
BEFORE DELETE ON cinema_user
FOR EACH ROW
DECLARE
  v_count NUMBER;
BEGIN
  SELECT COUNT(*) INTO v_count
  FROM ticket
  WHERE user_id = :OLD.user_id;

  IF v_count > 0 THEN
    RAISE_APPLICATION_ERROR(-20000, 'User has booked tickets and cannot be deleted!');
  END IF;
END;
/


BEGIN
    DELETE FROM cinema_user
    WHERE user_id = 'U0001';
END;
/
```



```
BEGIN
*
ERROR at line 1:
ORA-20000: User has booked tickets and cannot be deleted!
ORA-06512: at "CINEMABOOKING.TRG_PREVENT_USER_DELETE_WITH_TICKET", line 9
ORA-04088: error during execution of trigger 'CINEMABOOKING.TRG_PREVENT_USER_DELETE_WITH_TICKET'
ORA-06512: at line 2

https://docs.oracle.com/error-help/db/ora-20000/
```