MIDTERM

Our team:

Ayaulym Yesmakhanbet 210103323
Akniet Serik 210103340
Dilyara Zhaxylykova 210103322
Zhibek Dossymova 210103347

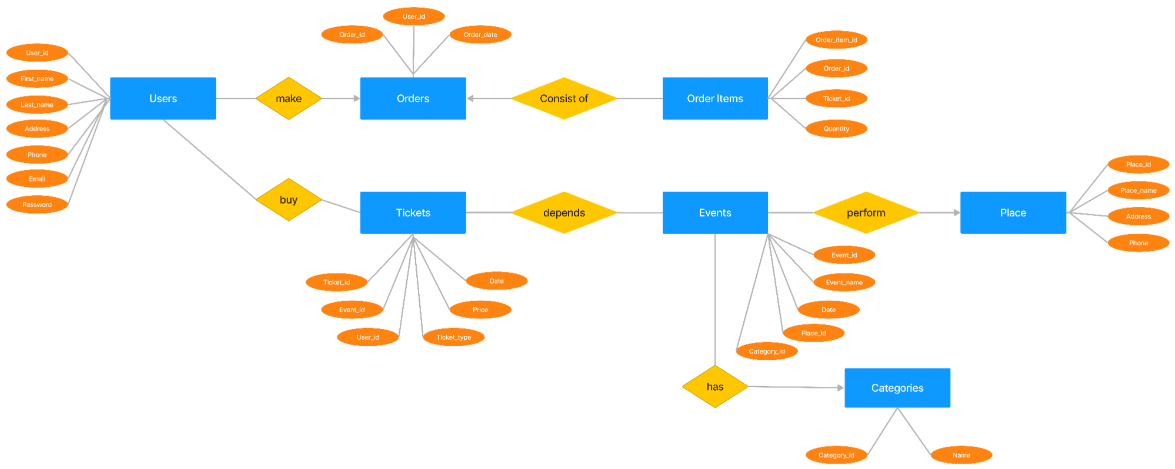
Introduction to the system:

Our **online ticket sales system** for cinema and various cultural and entertainment events.
A convenient way not to miss the long-awaited premiere.

Online ticket sales of cultural and sports events, cinema, theater, circus, concerts, excursions, museums and art galleries, children's, screenings, football, hockey and basketball matches of national teams, as well as international competitions.

By providing a multi-level service for the promotion and decomposition of tickets, this is an important link between the organizers and buyers of events.

ER diagram:



This scheme will allow the system to track all users, events, seats and tickets that are sold, which will allow for efficient data querying and management. Relationships between tables ensure the integrity and consistency of data and allow you to query data based on various criteria.

1) The Users table will store information about the users of the system, including their email address, password, first name, last name, address and phone number.

2) The Events table will store information about events, including the event name, description, date and time, as well as the ID of the place where the event will take place, as well as the ID of the event category.

3) The Place table will store information about the places, including the name, address and phone number.

4) The Categories table will store information about event categories such as music, sports, theater, etc.

5) The Tickets table will store information about tickets sold, including Event ID, User ID, ticket type, price, quantity and purchase date.

6) The Orders table will store information about orders made by users, including the order ID, User ID, and order date.

7) The Order Items table will store information about each item in each order, including the item ID in the order, the order ID, the ticket ID and the number of tickets ordered.

The database for an online ticket sales system with 7 tables has the following relationships:

1) The one-to-many relationship between the Place table and the Events table. One concert hall can host many events, but each event takes place in only one concert hall.

2) The one-to-many relationship between the Categories table and the Events table. Each event belongs to only one category, but each category can contain many events.

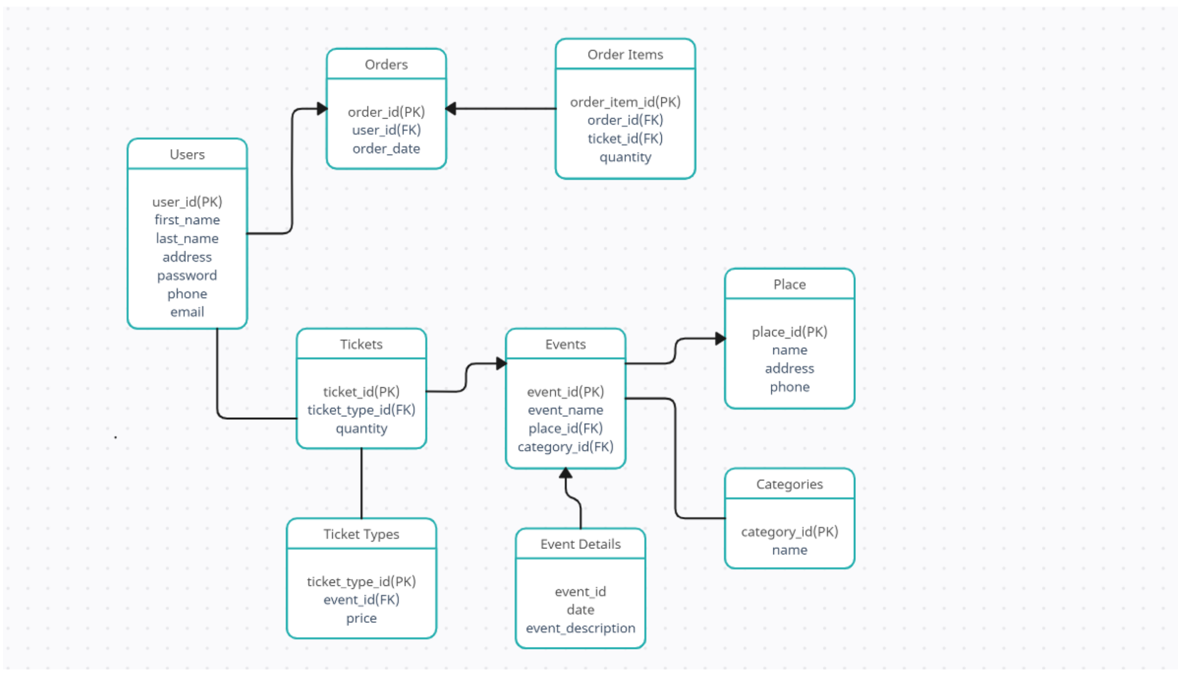
3) The many-to-many relationship between the Events and Users tables via the Tickets intermediate table. Each event can have many tickets sold, and each user can buy many tickets for different events.

4) The one-to-many relationship between the Users table and the Orders table. Each user can place multiple orders, but each order can only be placed by one user.

5) The one-to-many relationship between the Orders table and the Order Items table. Each order can contain multiple order items (i.e. multiple tickets), but each order item can belong to only one order.

Explanation of why the structure follows normal forms:

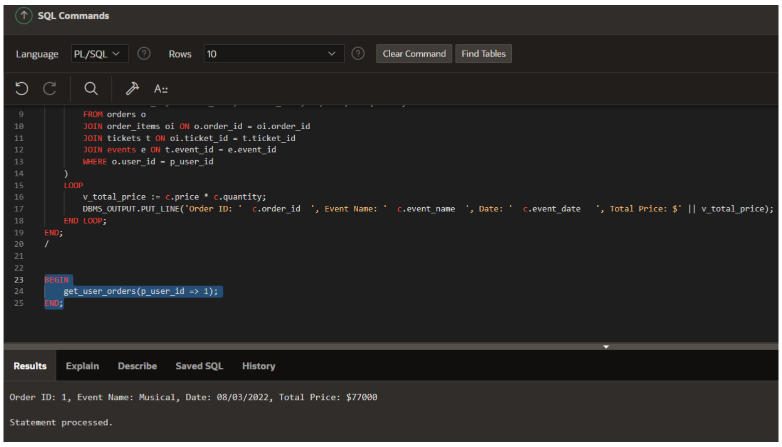
1) All tables are already in 1NF because each table has a primary key and all columns contain atomic values.
2) Normalize to 2NF:
   To achieve 2NF, we need to make sure that each non-key column in each table depends on the entire primary key. In the original schema, the Events table has a non-key column (date) that depends on only part of the primary key (event_id). We need to separate this column into its own table.
3) Normalize to 3NF:
   To achieve 3NF, we need to make sure that each non-key column in each table depends only on the primary key or other non-key columns. In the original schema, the Tickets table has a non-key column (price) that depends on the ticket_type column. We need to separate those columns into their own tables.



Explanation of the coding part:

1) Procedure

```
CREATE OR REPLACE PROCEDURE get_user_orders(
    p_user_id IN NUMBER
)
IS
    v_total_price NUMBER(10,2);
BEGIN
    FOR c IN (
        SELECT o.order_id, e.event_name, e.event_date, t.price, oi.quantity
        FROM orders o
        JOIN order_items oi ON o.order_id = oi.order_id
        JOIN tickets t ON oi.ticket_id = t.ticket_id
        JOIN events e ON t.event_id = e.event_id
        WHERE o.user_id = p_user_id
    )
    LOOP
        v_total_price := c.price * c.quantity;
        DBMS_OUTPUT.PUT_LINE('Order ID: '  c.order_id  ', Event Name: '  c.event_name  ', Date: '
c.event_date   ', Total Price: $' || v_total_price);
    END LOOP;
END;
/


BEGIN
    get_user_orders(p_user_id => 1);
END;
```

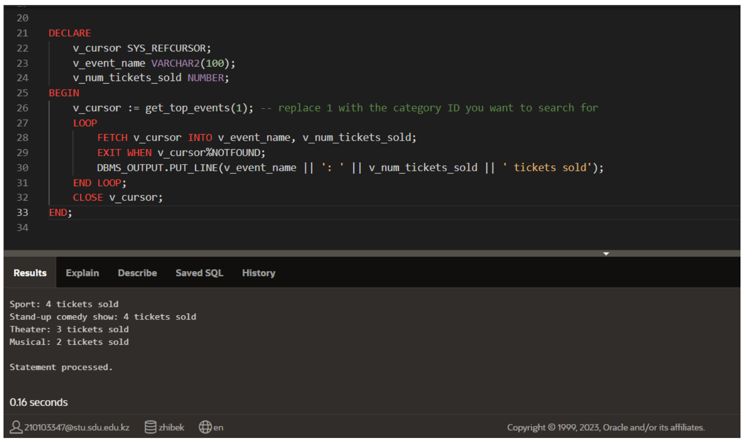


```
        FROM orders o
        JOIN order_items oi ON o.order_id = oi.order_id
        JOIN tickets t ON oi.ticket_id = t.ticket_id
        JOIN events e ON t.event_id = e.event_id
        WHERE o.user_id = p_user_id
    )
    LOOP
        v_total_price := c.price * c.quantity;
        DBMS_OUTPUT.PUT_LINE('Order ID: '  c.order_id  ', Event Name: '  c.event_name  ', Date: '  c.event_date   ', Total Price: $' || v_total_price);
    END LOOP;
END;
/


BEGIN
    get_user_orders(p_user_id => 1);
END;
```

```
Order ID: 1, Event Name: Musical, Date: 08/03/2022, Total Price: $77000

Statement processed.
```

2) Function

```
CREATE OR REPLACE FUNCTION get_top_events(
    p_category_id IN NUMBER
) RETURN SYS_REFCURSOR
IS
    v_cursor SYS_REFCURSOR;
BEGIN
    OPEN v_cursor FOR
        SELECT e.event_name, COUNT(t.ticket_id) AS num_tickets_sold
        FROM events e
        JOIN tickets t ON e.event_id = t.event_id
        JOIN categories c ON e.category_id = c.category_id
        WHERE c.category_id = p_category_id
        GROUP BY e.event_id, e.event_name
        ORDER BY num_tickets_sold DESC
        FETCH FIRST 5 ROWS ONLY;
    RETURN v_cursor;
END;


DECLARE
    v_cursor SYS_REFCURSOR;
    v_event_name VARCHAR2(100);
    v_num_tickets_sold NUMBER;
BEGIN
    v_cursor := get_top_events(1); -- replace 1 with the category ID you want to search for
    LOOP
        FETCH v_cursor INTO v_event_name, v_num_tickets_sold;
        EXIT WHEN v_cursor%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(v_event_name || ': ' || v_num_tickets_sold || ' tickets sold');
    END LOOP;
    CLOSE v_cursor;
END;
```



```
DECLARE
    v_cursor SYS_REFCURSOR;
    v_event_name VARCHAR2(100);
    v_num_tickets_sold NUMBER;
BEGIN
    v_cursor := get_top_events(1); -- replace 1 with the category ID you want to search for
    LOOP
        FETCH v_cursor INTO v_event_name, v_num_tickets_sold;
        EXIT WHEN v_cursor%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(v_event_name || ': ' || v_num_tickets_sold || ' tickets sold');
    END LOOP;
    CLOSE v_cursor;
END;
```

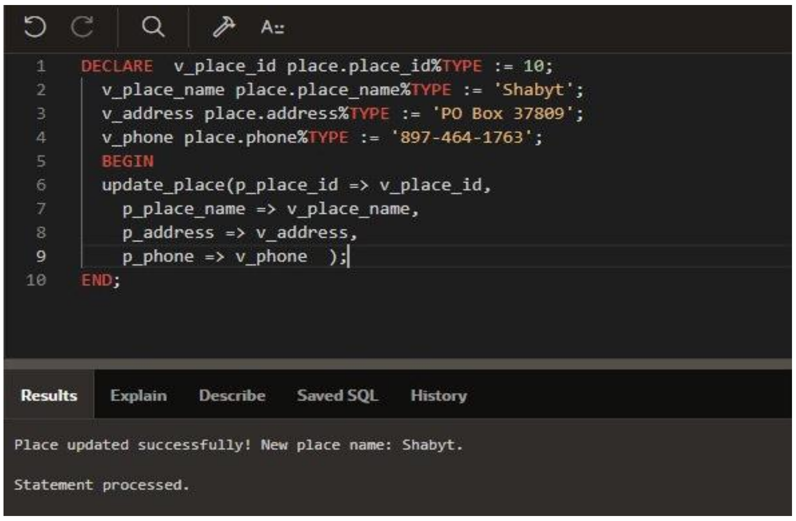
```
Sport: 4 tickets sold
Stand-up comedy show: 4 tickets sold
Theater: 3 tickets sold
Musical: 2 tickets sold

Statement processed.
```

0.16 seconds

210103347@stu.sdu.edu.kz zhibek en Copyright © 1999, 2023, Oracle and/or its affiliates.

3) Procedures which uses SQL%ROWCOUNT to determine the number of rows affected

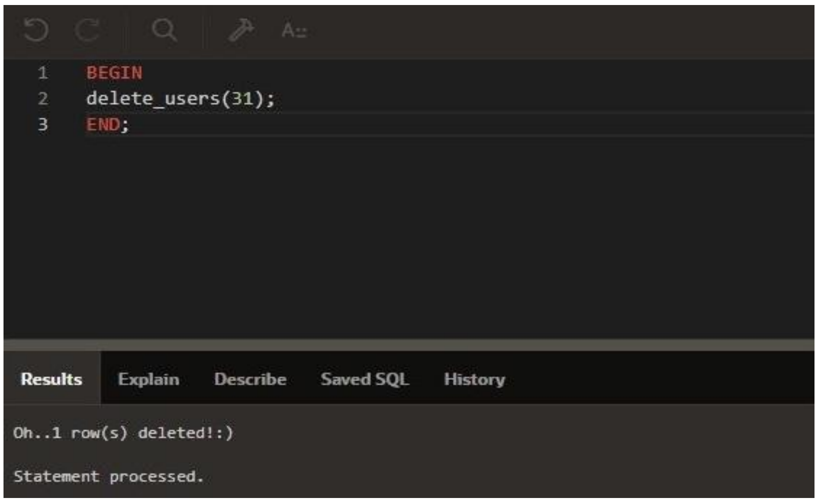
```
 create or replace PROCEDURE update_place(
   p_place_id IN place.place_id%TYPE,
   p_place_name IN place.place_name%TYPE,
   p_address in place.address%TYPE,
   p_phone in place.phone%TYPE
)
IS
BEGIN
   UPDATE place
   SET place_name = p_place_name
   WHERE place_id = p_place_id;

   IF SQL%ROWCOUNT = 1 THEN
      DBMS_OUTPUT.PUT_LINE('Place updated successfully! New place name: ' ||
p_place_name || '.');
   ELSIF SQL%ROWCOUNT = 0 THEN
      DBMS_OUTPUT.PUT_LINE('No place found with ID ' || p_place_id || '.');
   ELSE
      DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT || ' rows updated.');
   END IF;

   COMMIT;
EXCEPTION
   WHEN OTHERS THEN
      ROLLBACK;
      DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END;
```



```
DECLARE  v_place_id place.place_id%TYPE := 10;
  v_place_name place.place_name%TYPE := 'Shabyt';
  v_address place.address%TYPE := 'PO Box 37809';
  v_phone place.phone%TYPE := '897-464-1763';
  BEGIN
  update_place(p_place_id => v_place_id,
    p_place_name => v_place_name,
    p_address => v_address,
    p_phone => v_phone  );
END;
```

```
Place updated successfully! New place name: Shabyt.

Statement processed.
```

```
create or replace PROCEDURE delete_users
   (p_user_id IN Users.user_id%TYPE)
IS
BEGIN
   DELETE FROM Users WHERE user_id = p_user_id;

   IF SQL%ROWCOUNT > 0 THEN
      DBMS_OUTPUT.PUT_LINE('Oh..'|| SQL%ROWCOUNT || ' row(s) deleted!:)');
   ELSE
      DBMS_OUTPUT.PUT_LINE('No rows deleted.');
   END IF;
END;
```

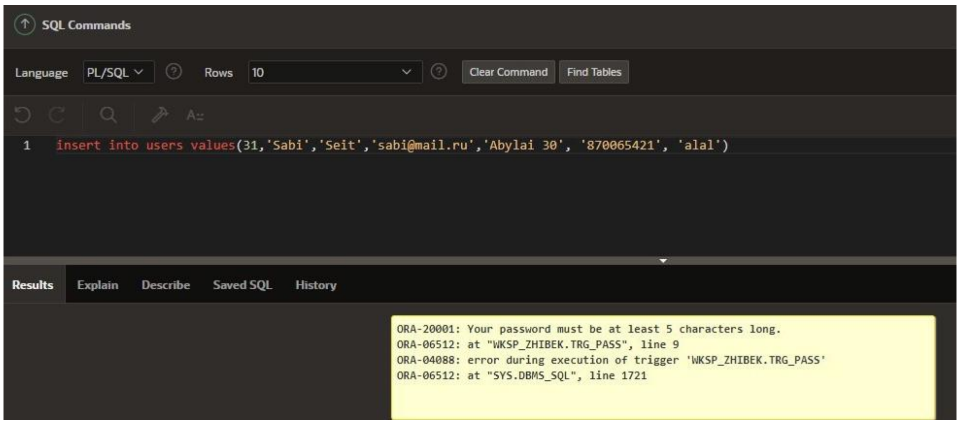


```
BEGIN
delete_users(31);
END;
```

```
Oh..1 row(s) deleted!:)

Statement processed.
```

4) Add user-defined exception which disallows to enter title of item to be less than 5 characters

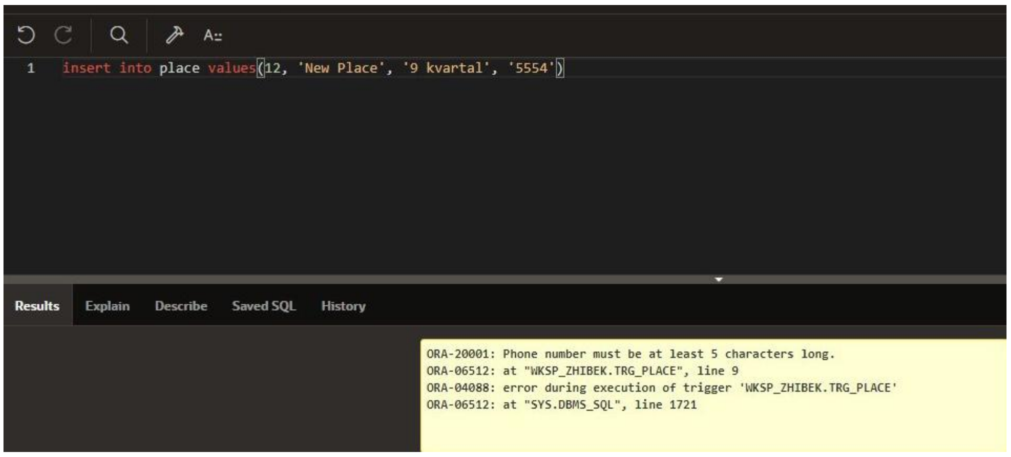
```
create or replace TRIGGER trg_pass
BEFORE INSERT OR UPDATE ON users
FOR EACH ROW
 DECLARE
   short_password EXCEPTION;
BEGIN
   IF LENGTH(:NEW.password) < 5 THEN
     RAISE short_password;
     END IF;
  EXCEPTION
  WHEN short_password THEN
   raise_application_error(-20001,'Your password must be at least 5 characters long.');
  WHEN others THEN
  raise_application_error(-20001,'Error');
END;
```



```
insert into users values(31,'Sabi','Seit','sabi@mail.ru','Abylai 30', '870065421', 'alal')
```

```
ORA-20001: Your password must be at least 5 characters long.
ORA-06512: at "WKSP_ZHIBEK.TRG_PASS", line 9
ORA-04088: error during execution of trigger 'WKSP_ZHIBEK.TRG_PASS'
ORA-06512: at "SYS.DBMS_SQL", line 1721
```

```
create or replace TRIGGER trg_Place
BEFORE INSERT ON Place
FOR EACH ROW
DECLARE
  short_phone EXCEPTION;
BEGIN
  IF LENGTH(:NEW.phone) < 5 THEN
    RAISE short_phone;
  END IF;
EXCEPTION
  WHEN short_phone THEN
    raise_application_error(-20001, 'Phone number must be at least 5 characters long.');
END;
```

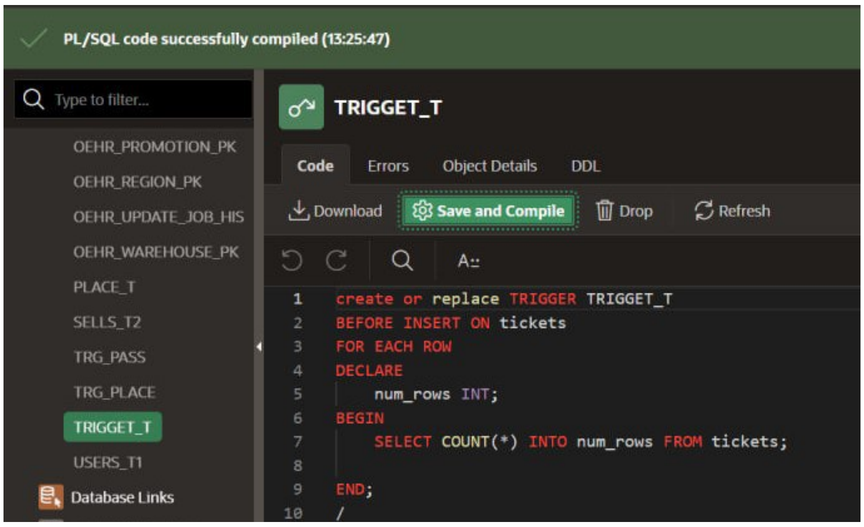


```
insert into place values(12, 'New Place', '9 kvartal', '5554')
```

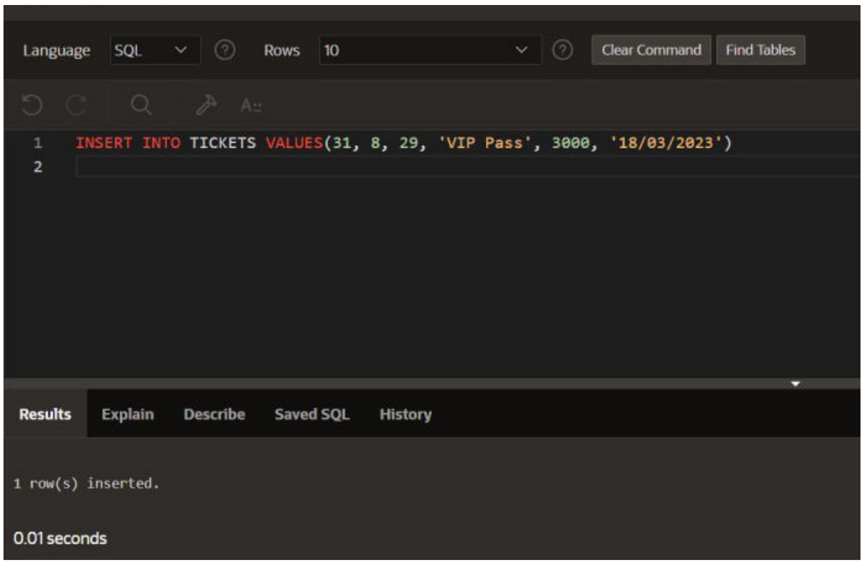
```
ORA-20001: Phone number must be at least 5 characters long.
ORA-06512: at "WKSP_ZHIBEK.TRG_PLACE", line 9
ORA-04088: error during execution of trigger 'WKSP_ZHIBEK.TRG_PLACE'
ORA-06512: at "SYS.DBMS_SQL", line 1721
```

5) Create a trigger before insert on any entity which will show the current number of rows in the table

create or replace TRIGGER TRIGGET_T
a)BEFORE INSERT ON tickets
FOR EACH ROWDECLARE
 num_rows INT;BEGIN
 SELECT COUNT(*) INTO num_rows FROM tickets;
END;
проверка
INSERT INTO TICKETS VALUES(31, 8, 29, 'VIP Pass', 3000, '18/03/2023')declare cou number;
begin SELECT COUNT(*) INTO cou FROM tickets;
 dbms_output.put_line(cou);end;

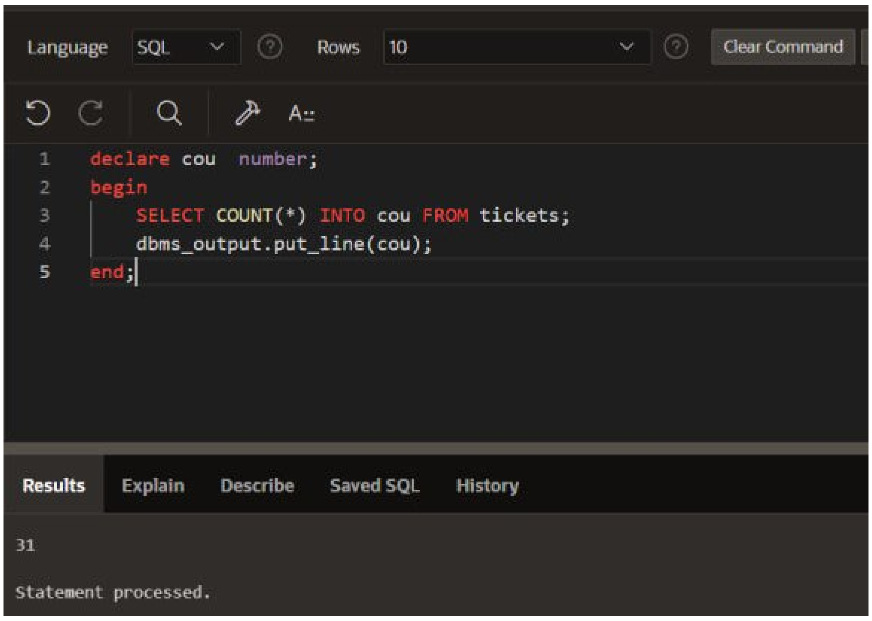

PL/SQL code successfully compiled (13:25:47)

```
create or replace TRIGGER TRIGGET_T
BEFORE INSERT ON tickets
FOR EACH ROW
DECLARE
    num_rows INT;
BEGIN
    SELECT COUNT(*) INTO num_rows FROM tickets;

END;
/
```



```
INSERT INTO TICKETS VALUES(31, 8, 29, 'VIP Pass', 3000, '18/03/2023')
```

```
1 row(s) inserted.

0.01 seconds
```

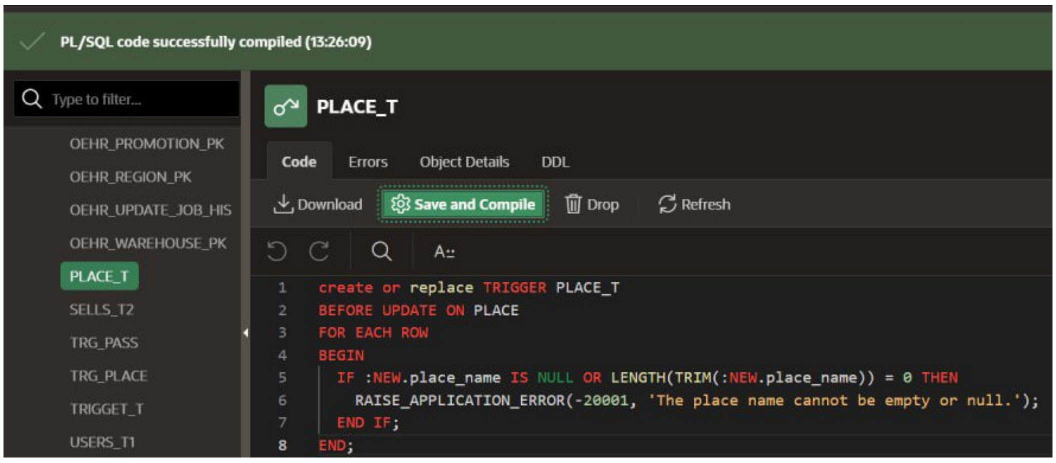


```
declare cou  number;
begin
    SELECT COUNT(*) INTO cou FROM tickets;
    dbms_output.put_line(cou);
end;
```

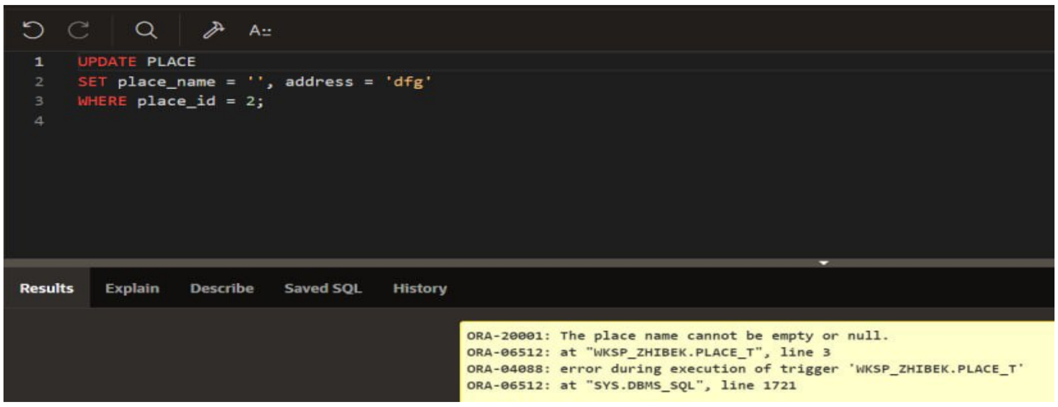
```
31

Statement processed.
```

b)CREATE OR REPLACE TRIGGER PLACE_TBEFORE UPDATE ON PLACE
FOR EACH ROWBEGIN
  IF :NEW.place_name IS NULL OR LENGTH(TRIM(:NEW.place_name)) = 0 THEN
RAISE_APPLICATION_ERROR(-20001, 'The place name cannot be empty or null.');
  END IF;END;

UPDATE PLACESET place_name = '', address = 'dfg'
WHERE place_id = 2;


PL/SQL code successfully compiled (13:26:09)

```
create or replace TRIGGER PLACE_T
BEFORE UPDATE ON PLACE
FOR EACH ROW
BEGIN
  IF :NEW.place_name IS NULL OR LENGTH(TRIM(:NEW.place_name)) = 0 THEN
    RAISE_APPLICATION_ERROR(-20001, 'The place name cannot be empty or null.');
  END IF;
END;
```



```
UPDATE PLACE
SET place_name = '', address = 'dfg'
WHERE place_id = 2;
```

```
ORA-20001: The place name cannot be empty or null.
ORA-06512: at "WKSP_ZHIBEK.PLACE_T", line 3
ORA-04088: error during execution of trigger 'WKSP_ZHIBEK.PLACE_T'
ORA-06512: at "SYS.DBMS_SQL", line 1721
```