

# MIDTERM

## Our team:

Ayaulym Yesmakhanbet 210103323

Akniet Serik 210103340

Dilyara Zhaxylykova 210103322

Zhibek Dossymova 210103347

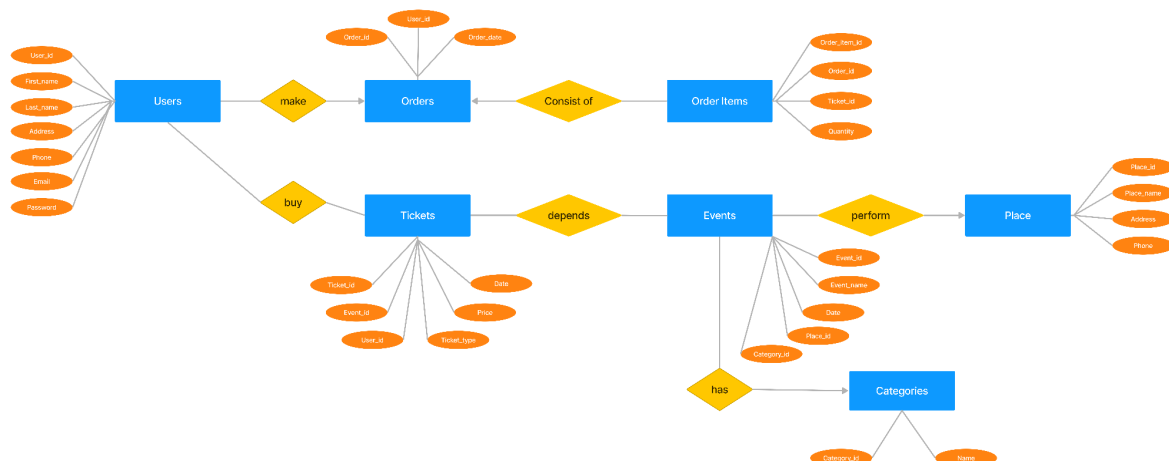
## Introduction to the system:

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.

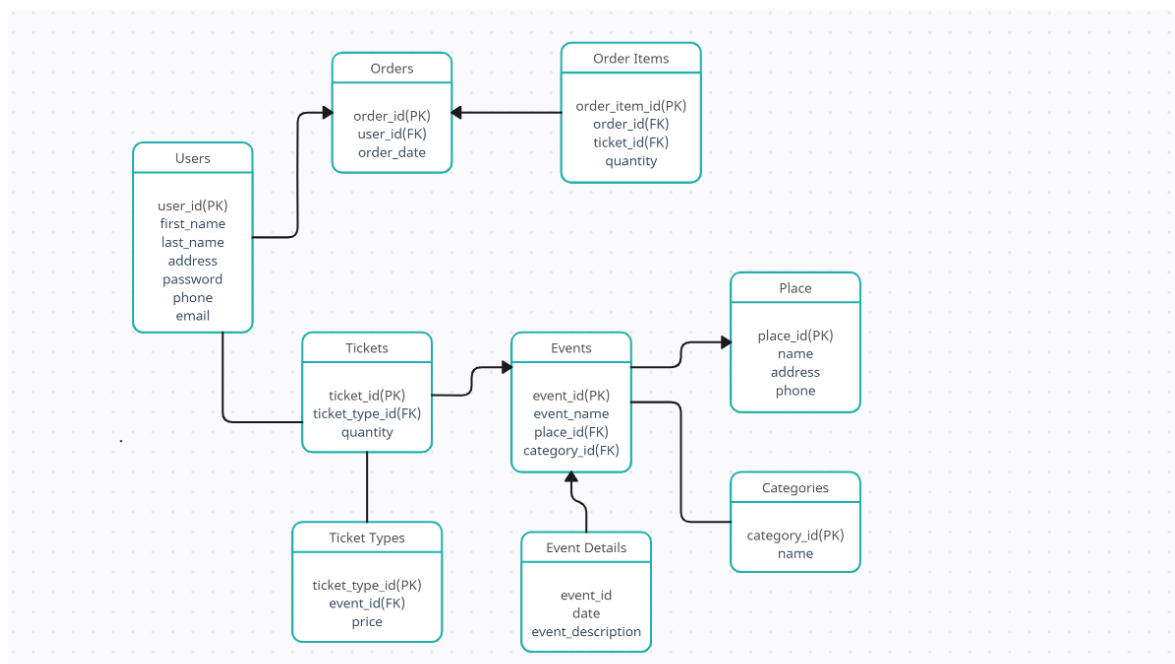
- 1) The Users table will store information about the users of the system, including their email address, password, first name, last name, address and phone number.
- 2) The Events table will store information about events, including the event name, description, date and time, as well as the ID of the 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:

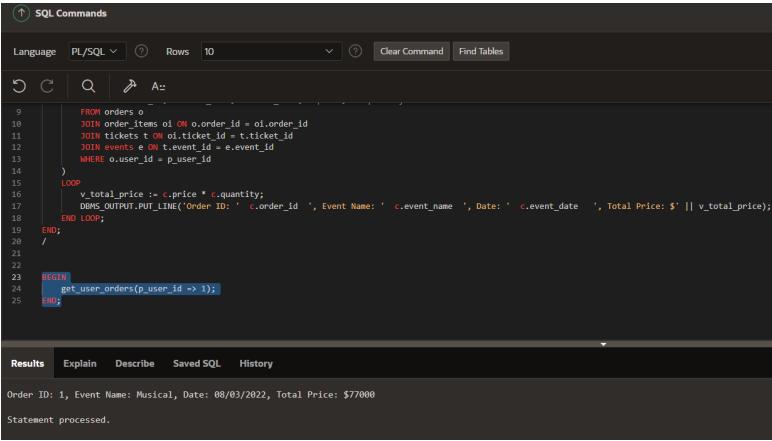
- 1) All tables are already in 1NF because each table has a primary key and all columns contain atomic values.
- 2) Normalize to 2NF:  
To achieve 2NF, we need to make sure that each non-key column in each table depends on the entire primary key. In the original schema, the Events table has a non-key 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;
```



The screenshot shows a SQL IDE interface with a dark theme. The top bar indicates 'SQL Commands' and 'PL/SQL' language. The main editor displays the PL/SQL code for the procedure, with line numbers 9 through 25. The code is as follows:

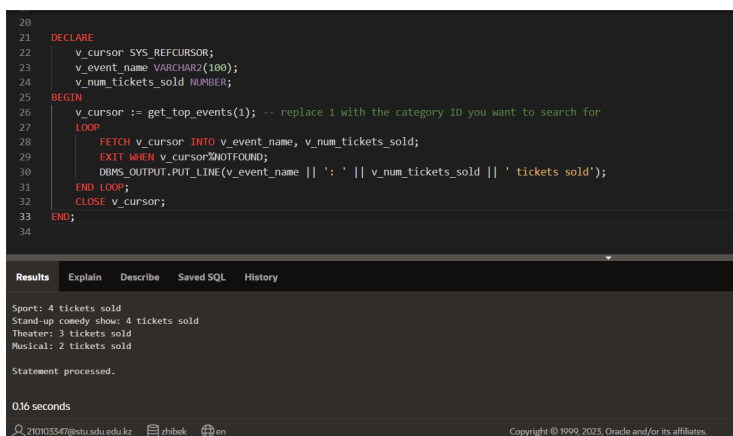
```
9      FROM orders o  
10     JOIN order_items oi ON o.order_id = oi.order_id  
11     JOIN tickets t ON oi.ticket_id = t.ticket_id  
12     JOIN events e ON t.event_id = e.event_id  
13     WHERE o.user_id = p_user_id  
14   )  
15   LOOP  
16     v_total_price := c.price * c.quantity;  
17     DBMS_OUTPUT.PUT_LINE('Order ID: ' || c.order_id || ', Event Name: ' || c.event_name || ', Date: ' || c.event_date || ', Total Price: $' || v_total_price);  
18   END LOOP;  
19 END;  
20 /  
21  
22  
23 BEGIN  
24   get_user_orders(p_user_id => 1);  
25 END;
```

Below the editor, the 'Results' tab is active, showing the output of the procedure:

```
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;
```



```
20  
21 DECLARE  
22     v_cursor SYS_REFCURSOR;  
23     v_event_name VARCHAR2(100);  
24     v_num_tickets_sold NUMBER;  
25 BEGIN  
26     v_cursor := get_top_events(1); -- replace 1 with the category ID you want to search for  
27     LOOP  
28         FETCH v_cursor INTO v_event_name, v_num_tickets_sold;  
29         EXIT WHEN v_cursor%NOTFOUND;  
30         DBMS_OUTPUT.PUT_LINE(v_event_name || ': ' || v_num_tickets_sold || ' tickets sold');  
31     END LOOP;  
32     CLOSE v_cursor;  
33 END;  
34
```

Results Explain Describe Saved SQL History

Sport: 4 tickets sold  
Stand-up comedy show: 4 tickets sold  
Theater: 3 tickets sold  
Musical: 2 tickets sold  
Statement processed.

0.16 seconds

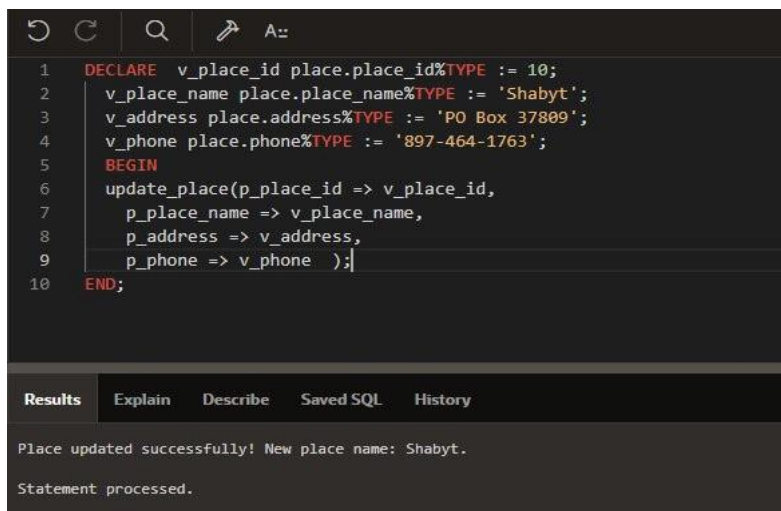
210103547@stu.sdu.edu.kz zhibek en Copyright © 1999, 2023, Oracle and/or its affiliates.

### 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;
```



The screenshot displays a SQL IDE interface. The top section shows a PL/SQL block with a procedure call: `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;`. The bottom section, titled 'Results', shows the output: 'Place updated successfully! New place name: Shabyt.' and 'Statement processed.'

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

**Results** Explain Describe Saved SQL History

Place updated successfully! New place name: Shabyt.

Statement processed.

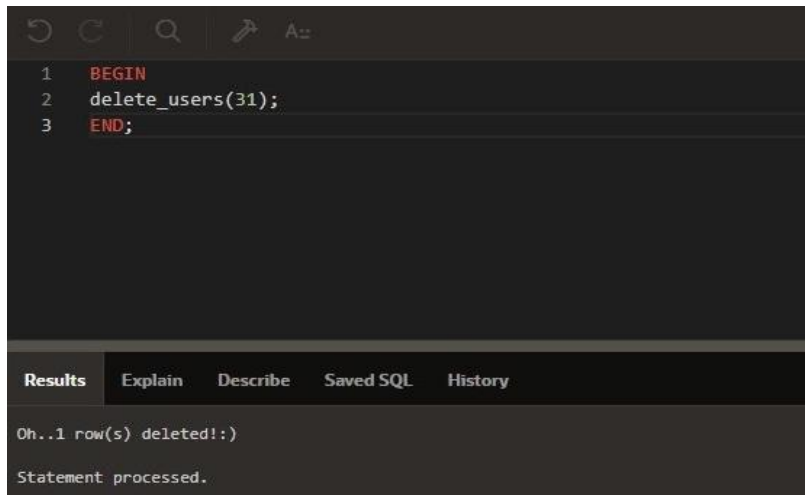
```

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;
```



The screenshot shows a SQL IDE with a dark theme. The top toolbar includes icons for undo, redo, search, and execution. The main editor area contains the following SQL code:

```

1  BEGIN
2  delete_users(31);
3  END;
```

Below the editor is a tabbed interface with 'Results', 'Explain', 'Describe', 'Saved SQL', and 'History'. The 'Results' tab is active, displaying the output of the execution:

```

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

At the bottom, a status bar indicates 'Statement processed.'

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

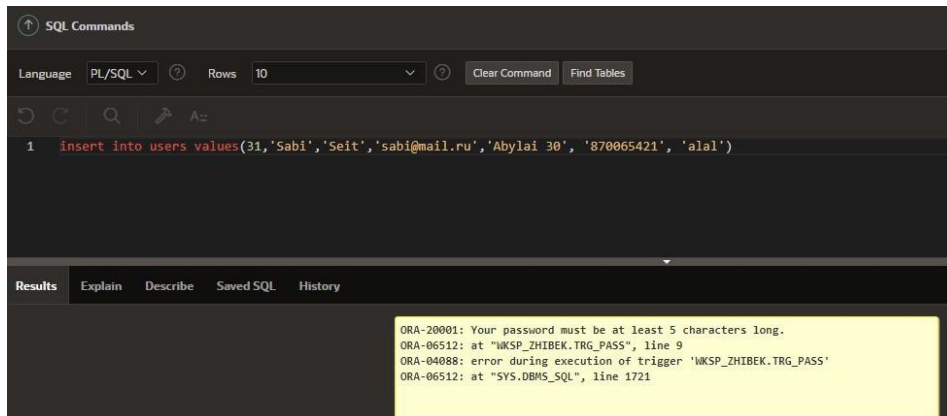
```

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;
```



SQL Commands

Language: PL/SQL Rows: 10 Clear Command Find Tables

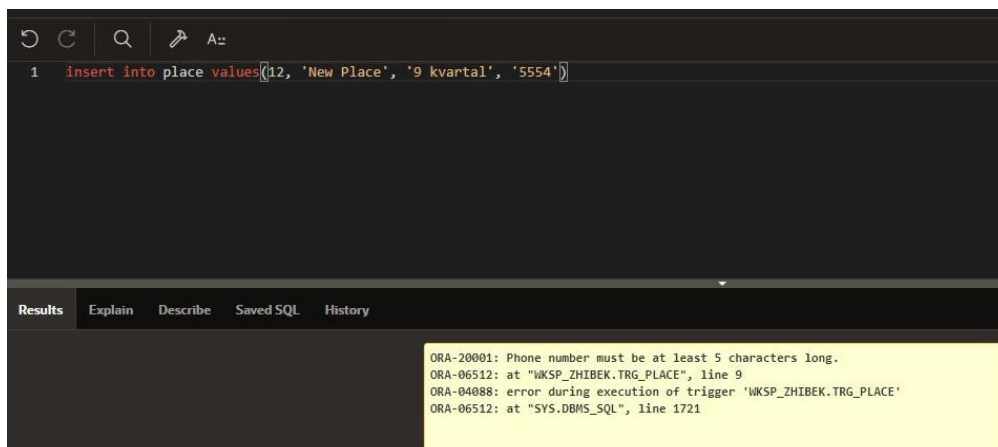
```
1 insert into users values(31,'Sabi','Seit','sabi@mail.ru','Abylai 30', '870065421', 'alal')
```

Results Explain Describe Saved SQL History

```
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;



SQL Commands

Language: PL/SQL Rows: 10 Clear Command Find Tables

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

Results Explain Describe Saved SQL History

```
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 ROW DECLARE

num\_rows INT; BEGIN

SELECT COUNT(\*) INTO num\_rows FROM tickets;

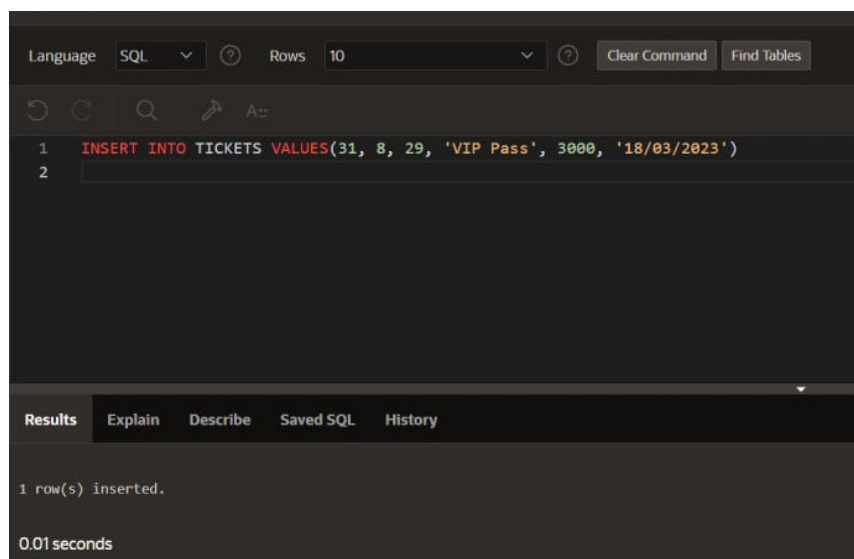
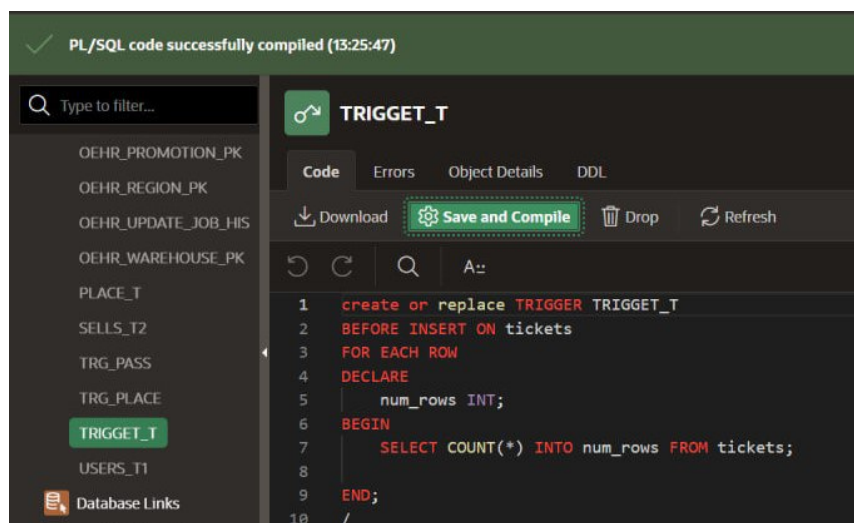
END;

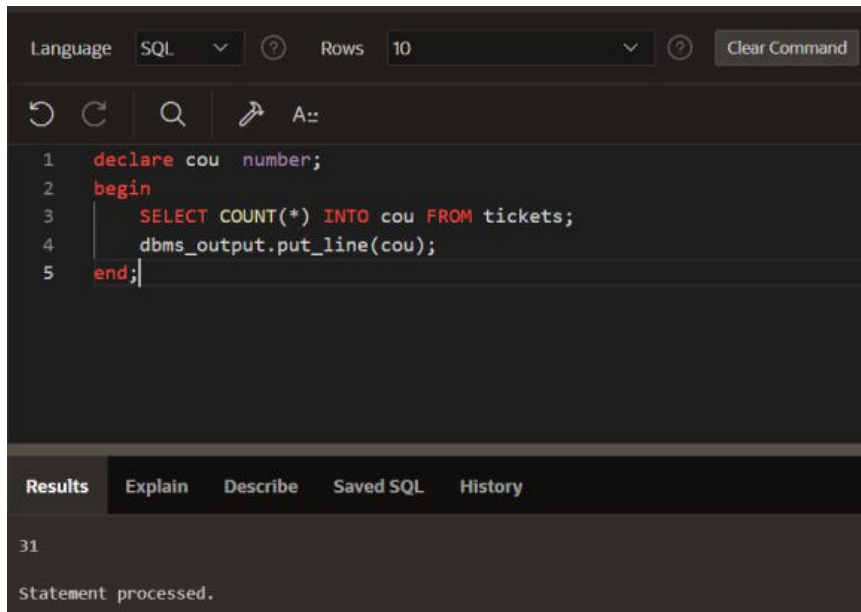
проверка

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;





The screenshot shows a SQL IDE interface. At the top, there's a toolbar with 'Language' set to 'SQL', 'Rows' set to '10', and a 'Clear Command' button. Below the toolbar is a command area with a search icon and a 'A::' prompt. The main area contains a PL/SQL block:

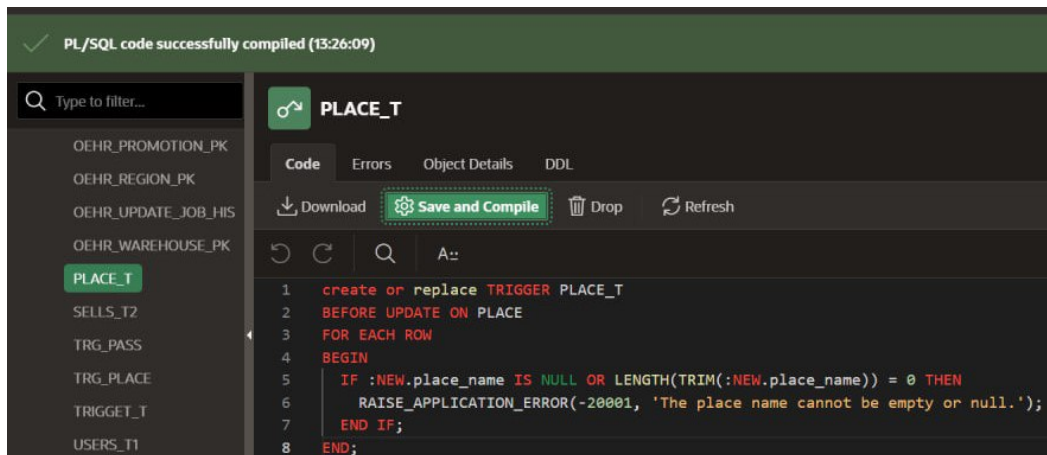
```
1 declare cou number;
2 begin
3     SELECT COUNT(*) INTO cou FROM tickets;
4     dbms_output.put_line(cou);
5 end;
```

At the bottom, there's a 'Results' tab with sub-tabs 'Explain', 'Describe', 'Saved SQL', and 'History'. The 'Results' tab is active, showing the text '31' and 'Statement processed.'

b)CREATE OR REPLACE TRIGGER PLACE\_TB  
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 PLACES place\_name = '', address = 'dfg'  
WHERE place\_id = 2;



The screenshot shows a SQL IDE interface. At the top, there's a green banner that says 'PL/SQL code successfully compiled (13:26:09)'. Below the banner is a sidebar with a search bar and a list of objects. The 'PLACE\_T' object is selected. The main area shows the code for the trigger:

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

A::

1 UPDATE PLACE

2 SET place\_name = '', address = 'dfg'

3 WHERE place\_id = 2;

4

Results

Explain

Describe

Saved SQL

History

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