DBI Protokoll – Elias, Rasti

Content:

Working Plan

Tables der DB; Rasti, Elias

Logging Table + Logging trigger; Rasti

Other 2 triggers - Elias

Tests; Rasti

User setup, inserts - Elias

Use case 1 - 10; Elias

Use case 11, 12; Rasti

Documentation; Rasti + Elias

SETUP:

- Open the setup folder
- Execute the user.sql file as admin of your oracle sql database
- Connect to the newly created user:
    - Username: casino
    - Password: oracle
- Execute the table.sql as the casino user
- Execute the inserts.sql as the casino user

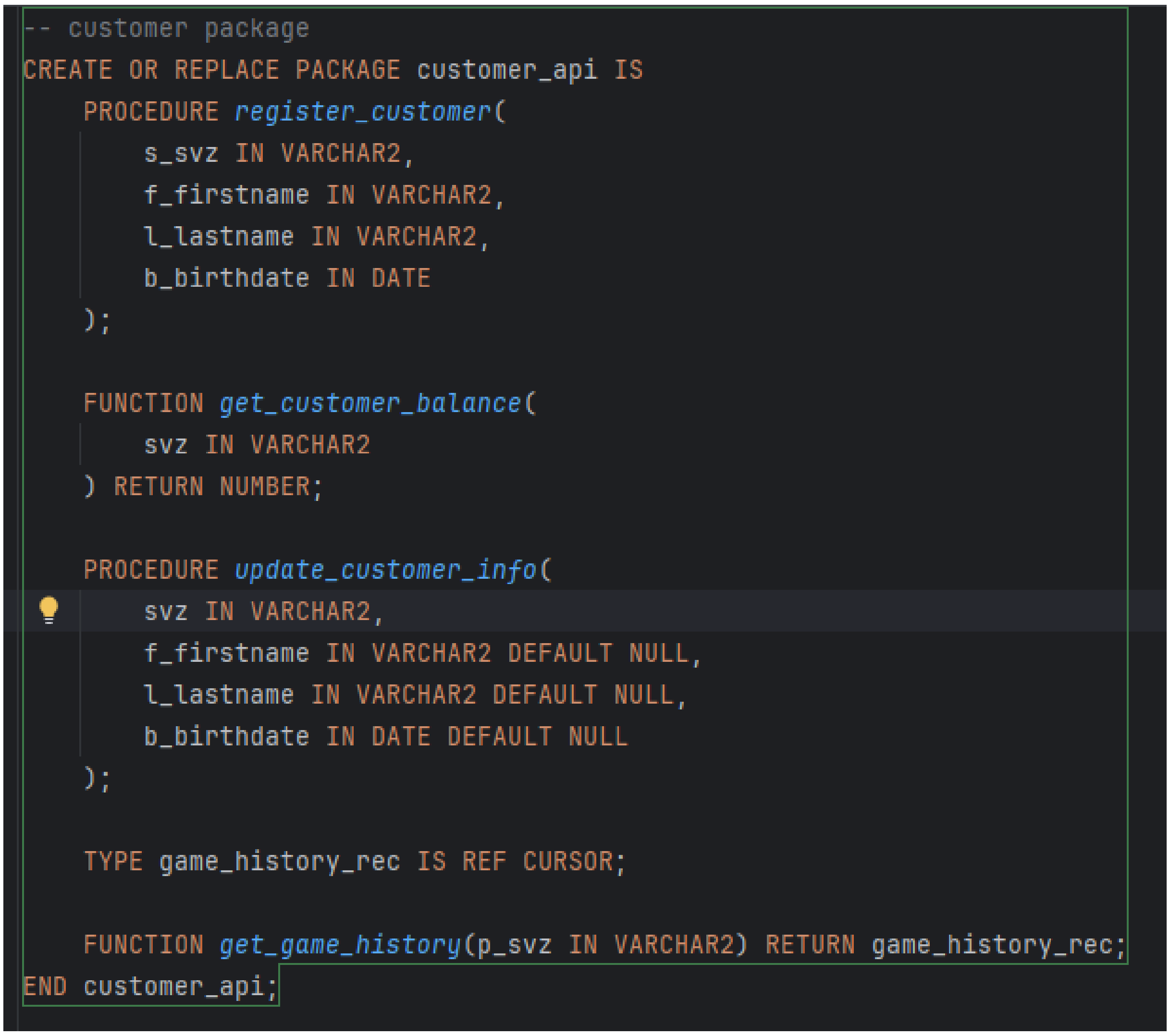
The drop.sql file is there if all tables need to be dropped

- Get back into the project_2025 folder
- Execute the log_trg.sql file as casino user
- Execute the trigger.sql file as casino user
- Go into the package folder
- Execute all the following sql files
    - Customer.sql
    - Game.sql
    - Game_session.sql
    - Personal.sql
    - Reports.sql
    - Table.sql

The Database should be setup now.

Use Cases:

Customer Use Case 1, 2, 3, 4 (Register New Customer, Get Customer Balance, Update Customer Info, Get Customer Game History):

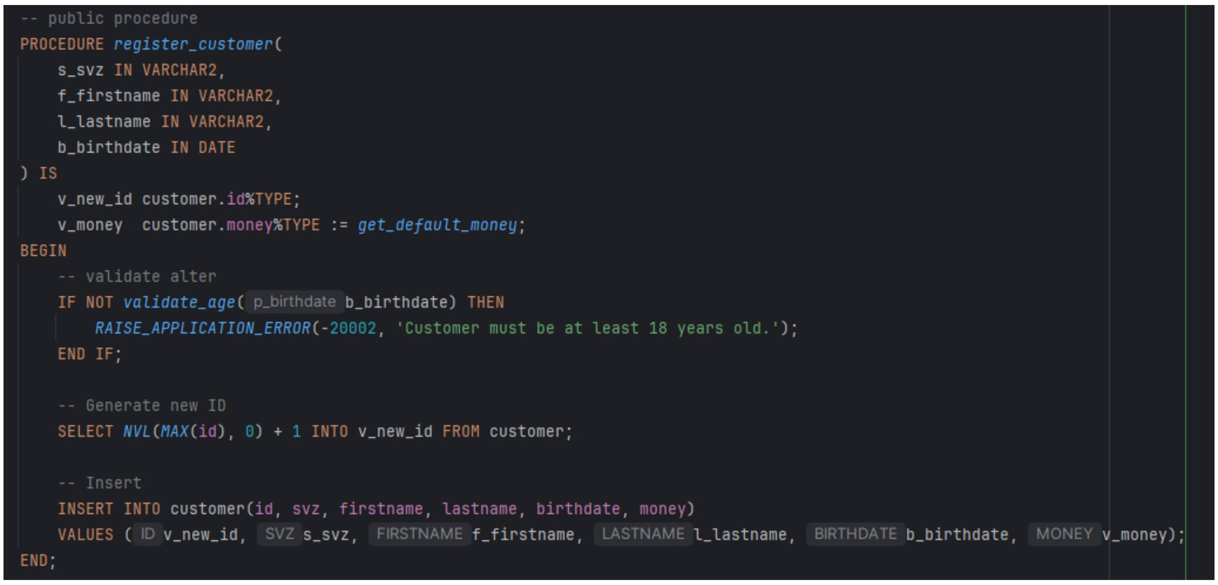


```
-- customer package
CREATE OR REPLACE PACKAGE customer_api IS
    PROCEDURE register_customer(
        s_svz IN VARCHAR2,
        f_firstname IN VARCHAR2,
        l_lastname IN VARCHAR2,
        b_birthdate IN DATE
    );

    FUNCTION get_customer_balance(
        svz IN VARCHAR2
    ) RETURN NUMBER;

    PROCEDURE update_customer_info(
        svz IN VARCHAR2,
        f_firstname IN VARCHAR2 DEFAULT NULL,
        l_lastname IN VARCHAR2 DEFAULT NULL,
        b_birthdate IN DATE DEFAULT NULL
    );

    TYPE game_history_rec IS REF CURSOR;

    FUNCTION get_game_history(p_svz IN VARCHAR2) RETURN game_history_rec;
END customer_api;
```

In the package folder in the customer.sql file there is the package for the first 4 use cases. The 4 public use cases are defined in the package.

In the body of the package there is the corresponding code and the private helper functions which are needed.

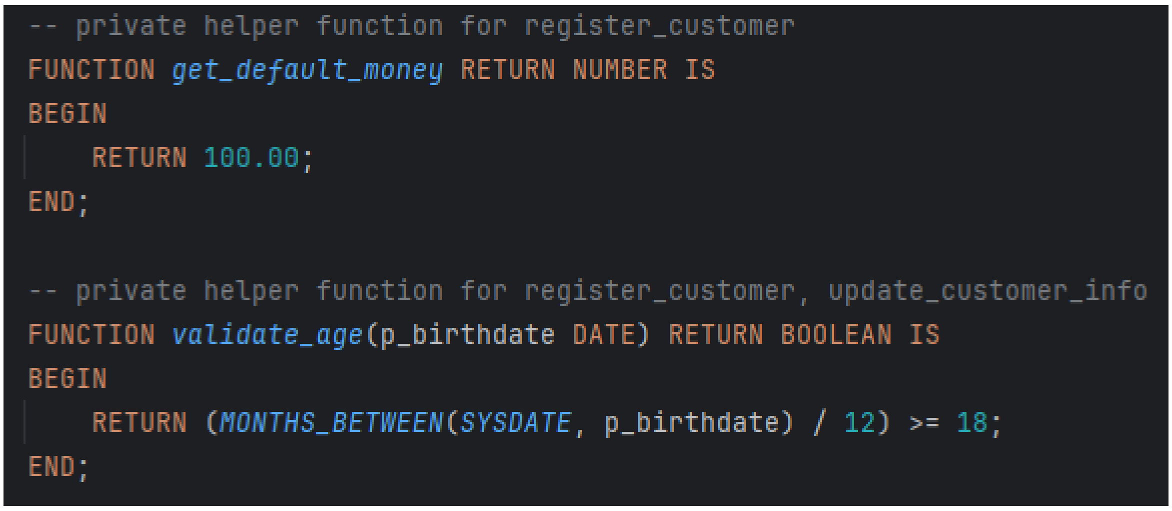
Use Case 1 (Register New Customer):



```sql
-- public procedure
PROCEDURE register_customer(
    s_svz IN VARCHAR2,
    f_firstname IN VARCHAR2,
    l_lastname IN VARCHAR2,
    b_birthdate IN DATE
) IS
    v_new_id customer.id%TYPE;
    v_money  customer.money%TYPE := get_default_money;
BEGIN
    -- validate alter
    IF NOT validate_age( p_birthdate b_birthdate) THEN
        RAISE_APPLICATION_ERROR(-20002, 'Customer must be at least 18 years old.');
    END IF;

    -- Generate new ID
    SELECT NVL(MAX(id), 0) + 1 INTO v_new_id FROM customer;

    -- Insert
    INSERT INTO customer(id, svz, firstname, lastname, birthdate, money)
    VALUES ( ID v_new_id,  SVZ s_svz,  FIRSTNAME f_firstname,  LASTNAME l_lastname,  BIRTHDATE b_birthdate,  MONEY v_money);
END;
```

Uses helper functions

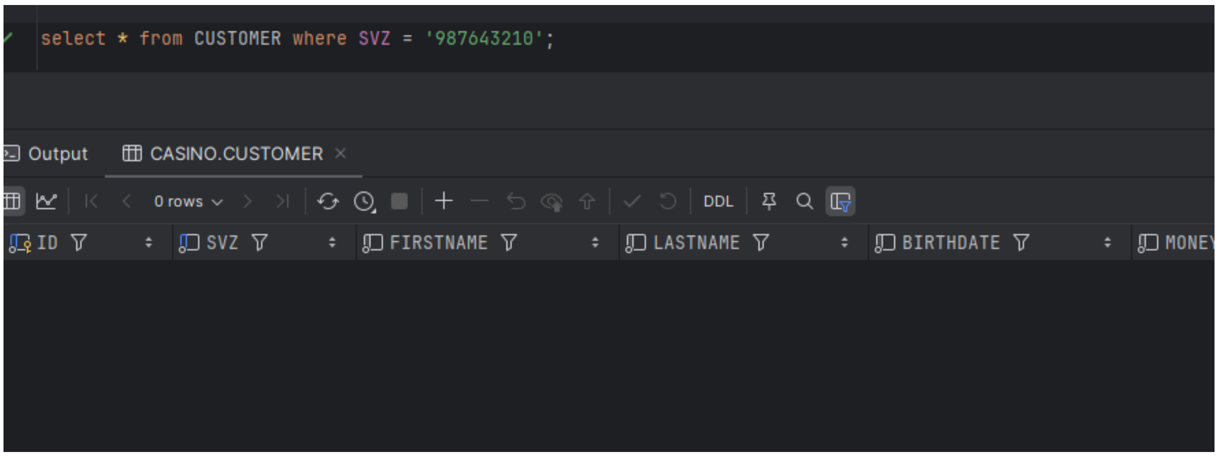
- Get default money and validate age



```sql
-- private helper function for register_customer
FUNCTION get_default_money RETURN NUMBER IS
BEGIN
    RETURN 100.00;
END;

-- private helper function for register_customer, update_customer_info
FUNCTION validate_age(p_birthdate DATE) RETURN BOOLEAN IS
BEGIN
    RETURN (MONTHS_BETWEEN(SYSDATE, p_birthdate) / 12) >= 18;
END;
```

Test for use case 1:

All the tests for the packages are in the package test folder and in the corresponding file.



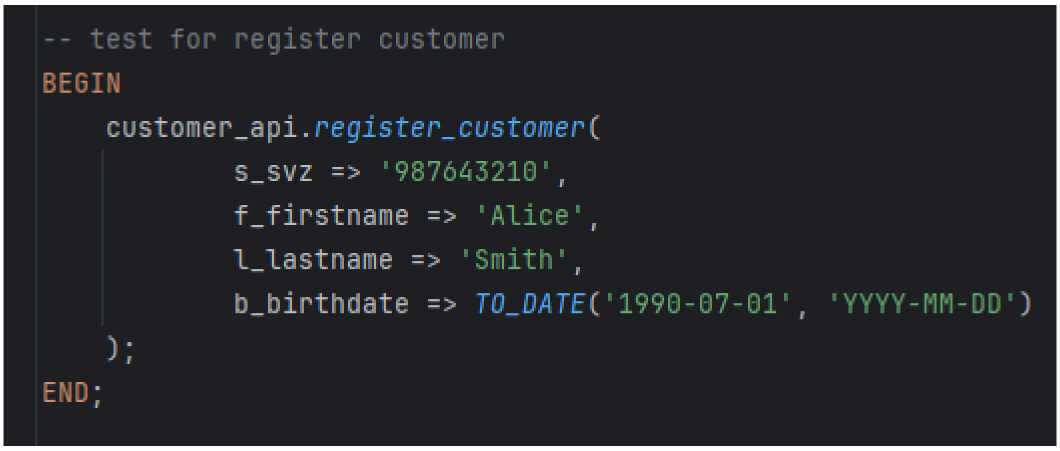
```
select * from CUSTOMER where SVZ = '987643210';
```

Output | CASINO.CUSTOMER

0 rows

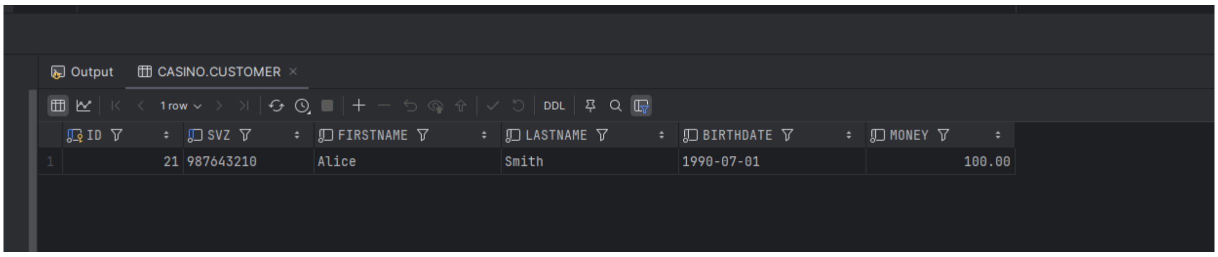
ID	SVZ	FIRSTNAME	LASTNAME	BIRTHDATE	MONEY

No data with the svz



```
-- test for register customer
BEGIN
    customer_api.register_customer(
            s_svz => '987643210',
            f_firstname => 'Alice',
            l_lastname => 'Smith',
            b_birthdate => TO_DATE('1990-07-01', 'YYYY-MM-DD')
    );
END;
```

After execution:

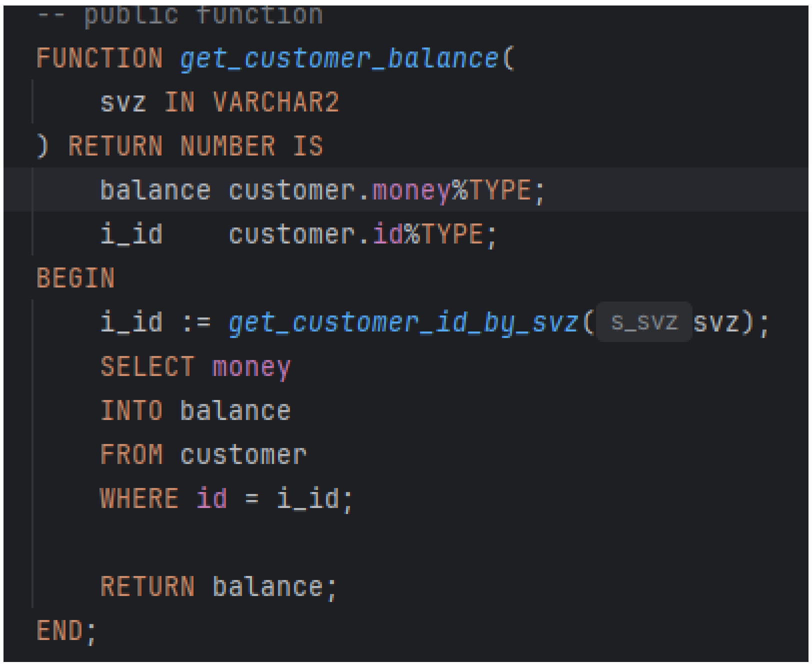


Output | CASINO.CUSTOMER

1 row

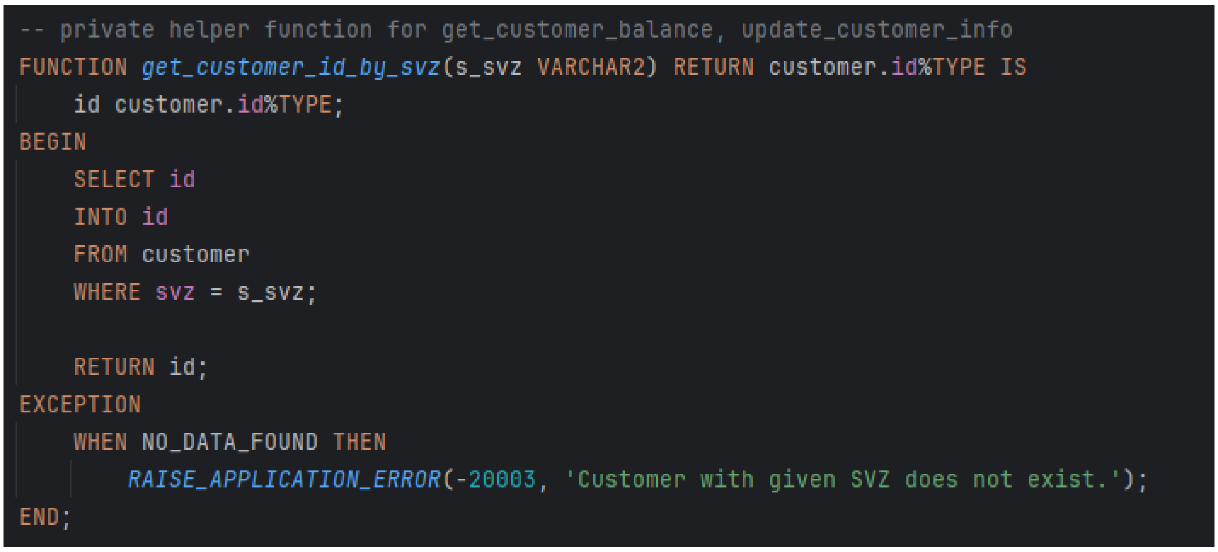
	ID	SVZ	FIRSTNAME	LASTNAME	BIRTHDATE	MONEY
1	21	987643210	Alice	Smith	1990-07-01	100.00

Use Case 2 (Get Customer Balance):



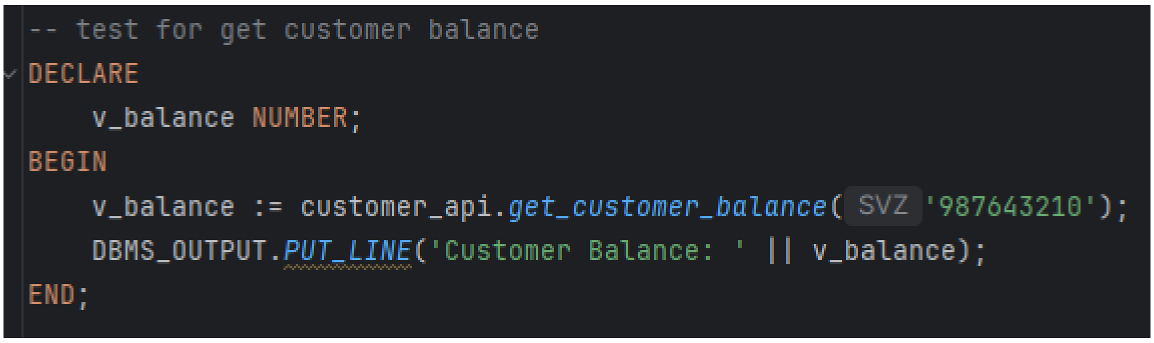
```
-- public function
FUNCTION get_customer_balance(
    svz IN VARCHAR2
) RETURN NUMBER IS
    balance customer.money%TYPE;
    i_id    customer.id%TYPE;
BEGIN
    i_id := get_customer_id_by_svz( s_svz svz);
    SELECT money
    INTO balance
    FROM customer
    WHERE id = i_id;

    RETURN balance;
END;
```

Helper function:



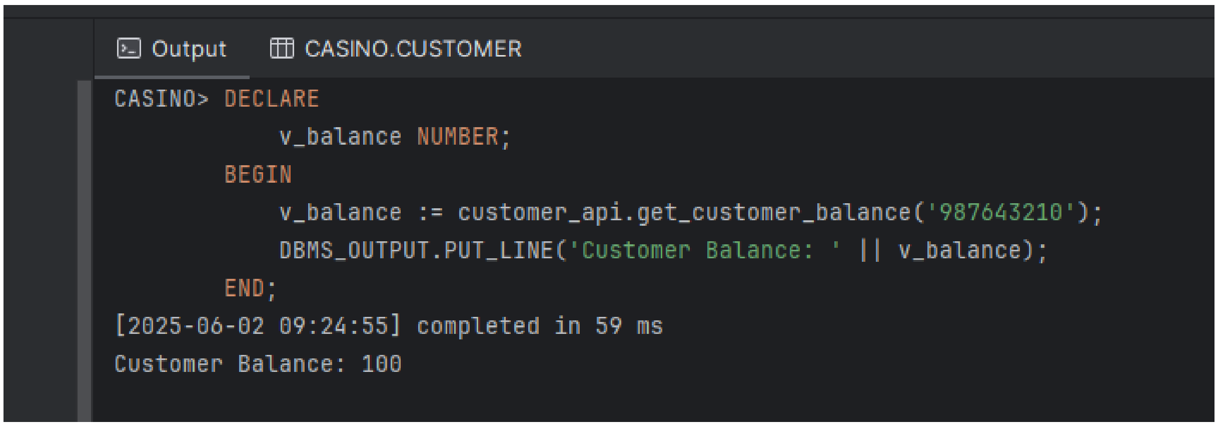
```
-- private helper function for get_customer_balance, update_customer_info
FUNCTION get_customer_id_by_svz(s_svz VARCHAR2) RETURN customer.id%TYPE IS
    id customer.id%TYPE;
BEGIN
    SELECT id
    INTO id
    FROM customer
    WHERE svz = s_svz;

    RETURN id;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RAISE_APPLICATION_ERROR(-20003, 'Customer with given SVZ does not exist.');
END;
```

Test for use case 2:



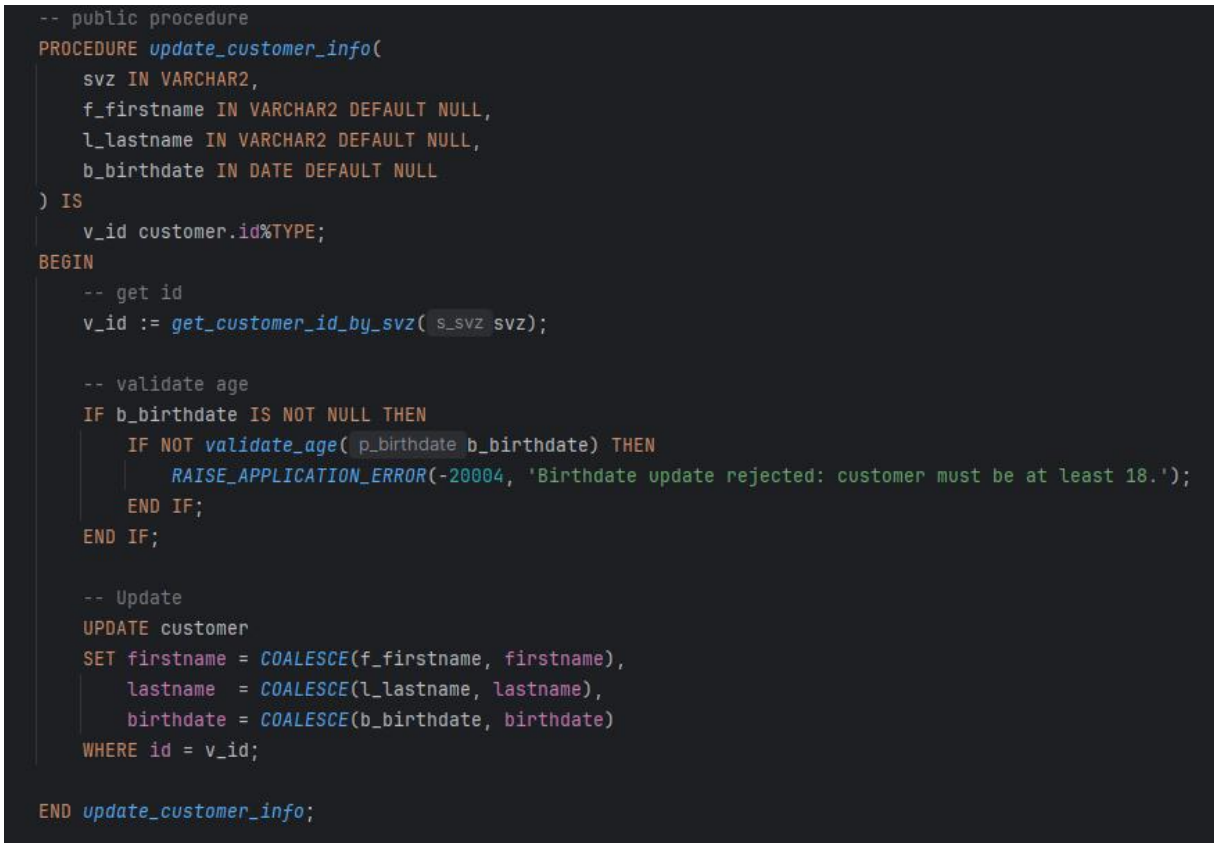
```
-- test for get customer balance
DECLARE
    v_balance NUMBER;
BEGIN
    v_balance := customer_api.get_customer_balance( SVZ '987643210');
    DBMS_OUTPUT.PUT_LINE('Customer Balance: ' || v_balance);
END;
```

Prints out the customers balance in this case the one of the created customer



```
CASINO> DECLARE
            v_balance NUMBER;
        BEGIN
            v_balance := customer_api.get_customer_balance('987643210');
            DBMS_OUTPUT.PUT_LINE('Customer Balance: ' || v_balance);
        END;
[2025-06-02 09:24:55] completed in 59 ms
Customer Balance: 100
```

Use Case 3 (Update Customer Info):

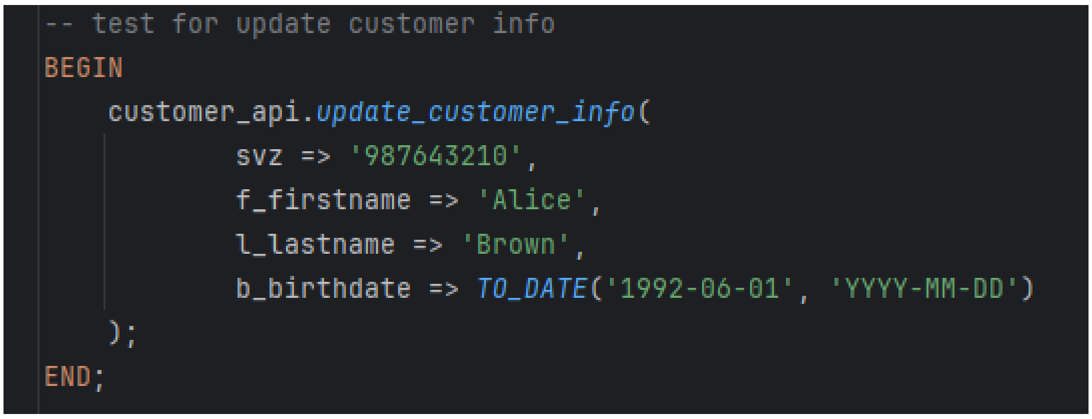


```
-- public procedure
PROCEDURE update_customer_info(
    svz IN VARCHAR2,
    f_firstname IN VARCHAR2 DEFAULT NULL,
    l_lastname IN VARCHAR2 DEFAULT NULL,
    b_birthdate IN DATE DEFAULT NULL
) IS
    v_id customer.id%TYPE;
BEGIN
    -- get id
    v_id := get_customer_id_by_svz( s_svz svz);

    -- validate age
    IF b_birthdate IS NOT NULL THEN
        IF NOT validate_age( p_birthdate b_birthdate) THEN
            RAISE_APPLICATION_ERROR(-20004, 'Birthdate update rejected: customer must be at least 18.');
        END IF;
    END IF;

    -- Update
    UPDATE customer
    SET firstname = COALESCE(f_firstname, firstname),
        lastname  = COALESCE(l_lastname, lastname),
        birthdate = COALESCE(b_birthdate, birthdate)
    WHERE id = v_id;

END update_customer_info;
```

Helper functions (same as the ones before):

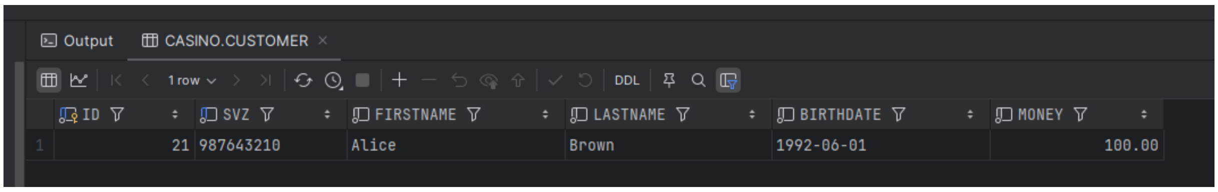
- Get customer id by svz
- Validate age

Test for use case 3:



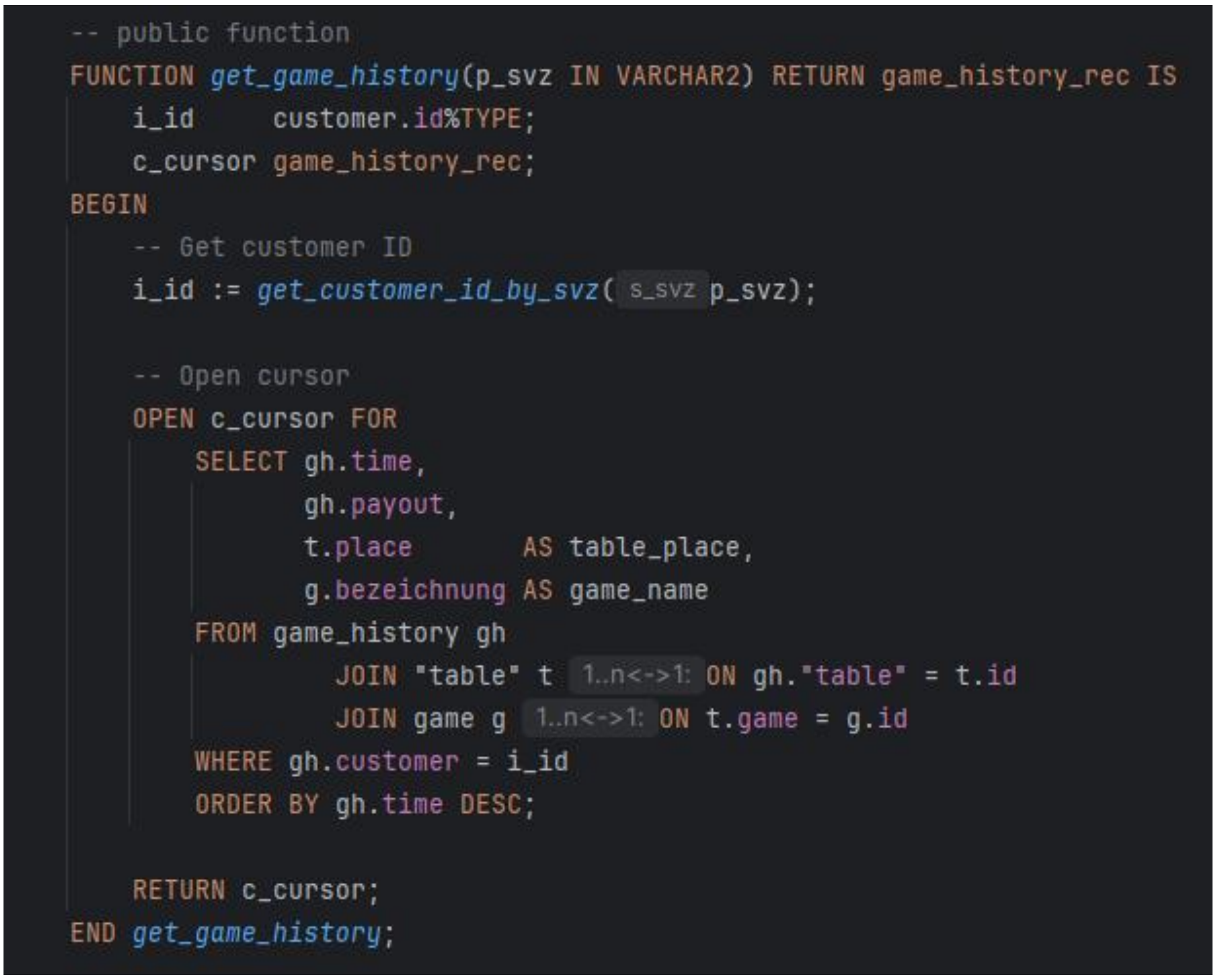
```
-- test for update customer info
BEGIN
    customer_api.update_customer_info(
            svz => '987643210',
            f_firstname => 'Alice',
            l_lastname => 'Brown',
            b_birthdate => TO_DATE('1992-06-01', 'YYYY-MM-DD')
    );
END;
```

After execution:



```
   ID  SVZ        FIRSTNAME  LASTNAME  BIRTHDATE   MONEY
1  21  987643210  Alice      Brown     1992-06-01  100.00
```

Use Case 4 (Get Customer Game History):

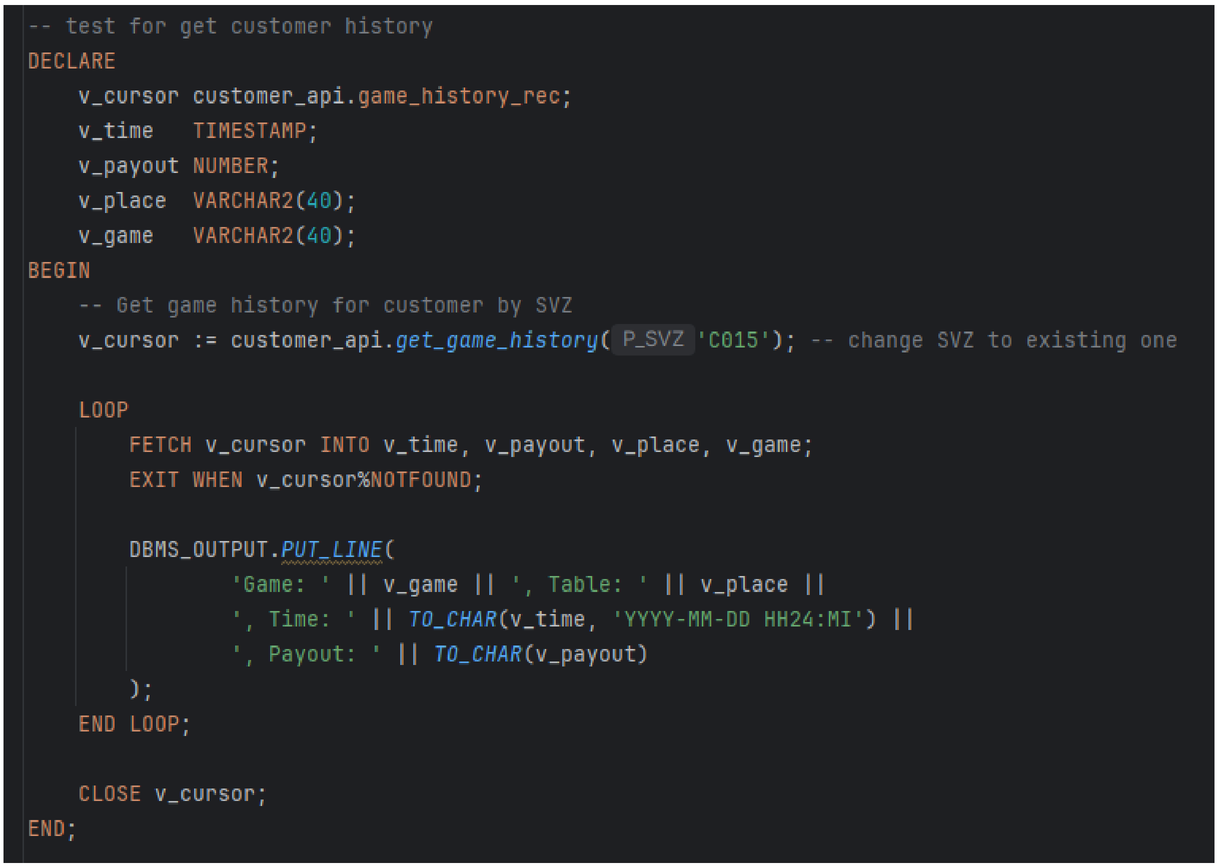


```
-- public function
FUNCTION get_game_history(p_svz IN VARCHAR2) RETURN game_history_rec IS
    i_id     customer.id%TYPE;
    c_cursor game_history_rec;
BEGIN
    -- Get customer ID
    i_id := get_customer_id_by_svz( s_svz p_svz);

    -- Open cursor
    OPEN c_cursor FOR
        SELECT gh.time,
               gh.payout,
               t.place       AS table_place,
               g.bezeichnung AS game_name
        FROM game_history gh
                 JOIN "table" t 1..n<->1: ON gh."table" = t.id
                 JOIN game g 1..n<->1: ON t.game = g.id
        WHERE gh.customer = i_id
        ORDER BY gh.time DESC;

    RETURN c_cursor;
END get_game_history;
```

Helper function:

- Get customer id by svz

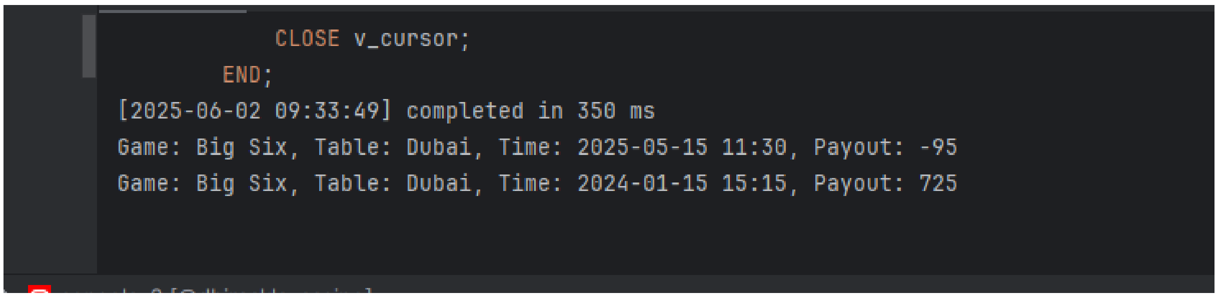
Test for use case 4:



```
-- test for get customer history
DECLARE
    v_cursor customer_api.game_history_rec;
    v_time   TIMESTAMP;
    v_payout NUMBER;
    v_place  VARCHAR2(40);
    v_game   VARCHAR2(40);
BEGIN
    -- Get game history for customer by SVZ
    v_cursor := customer_api.get_game_history( P_SVZ 'C015'); -- change SVZ to existing one

    LOOP
        FETCH v_cursor INTO v_time, v_payout, v_place, v_game;
        EXIT WHEN v_cursor%NOTFOUND;

        DBMS_OUTPUT.PUT_LINE(
                'Game: ' || v_game || ', Table: ' || v_place ||
                ', Time: ' || TO_CHAR(v_time, 'YYYY-MM-DD HH24:MI') ||
                ', Payout: ' || TO_CHAR(v_payout)
        );
    END LOOP;

    CLOSE v_cursor;
END;
```

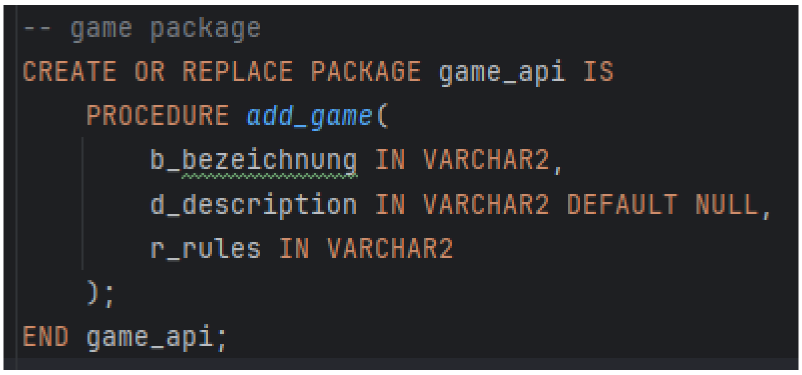
Because there are no records of our newly created user the user with the svz C015 is used.

All the played games get printed:

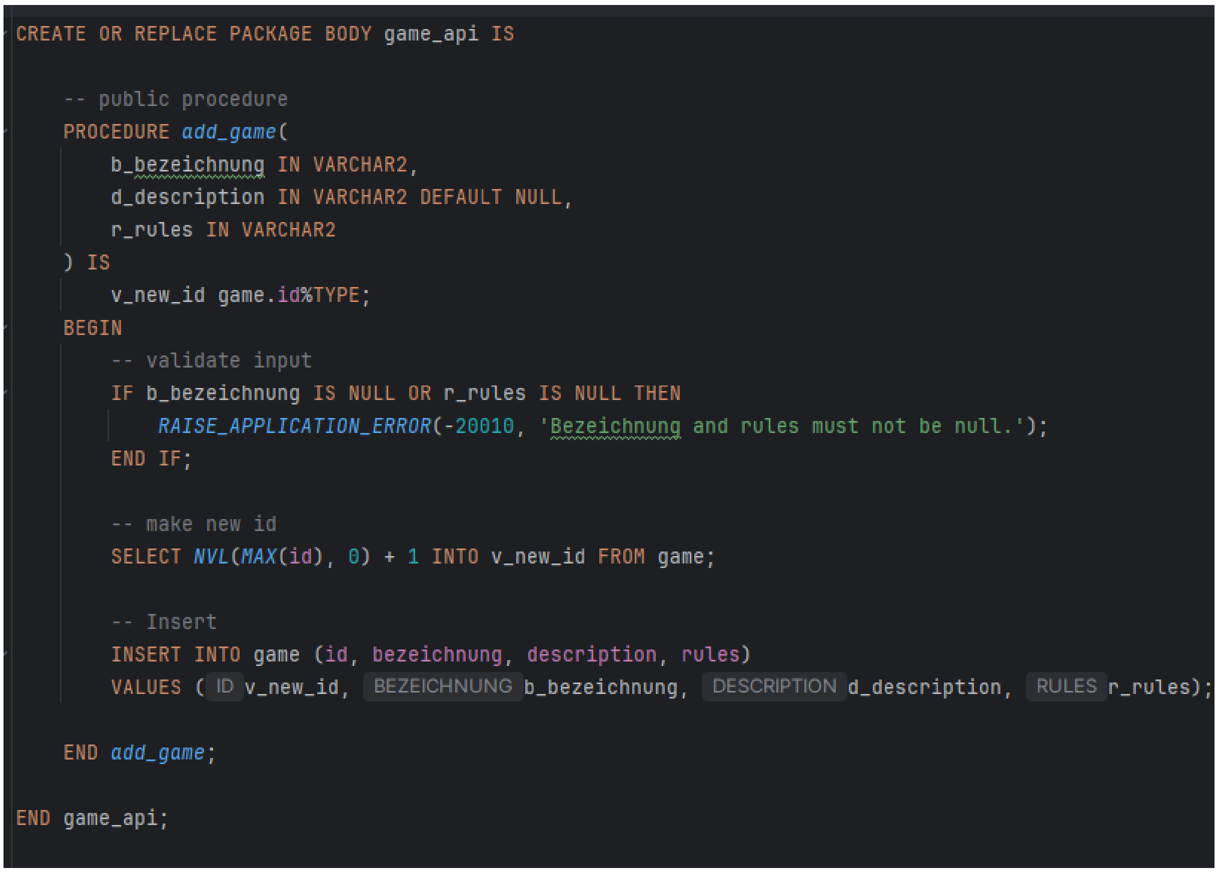


```
            CLOSE v_cursor;
        END;
[2025-06-02 09:33:49] completed in 350 ms
Game: Big Six, Table: Dubai, Time: 2025-05-15 11:30, Payout: -95
Game: Big Six, Table: Dubai, Time: 2024-01-15 15:15, Payout: 725
```

Game Use Case 5 (Add new Game):

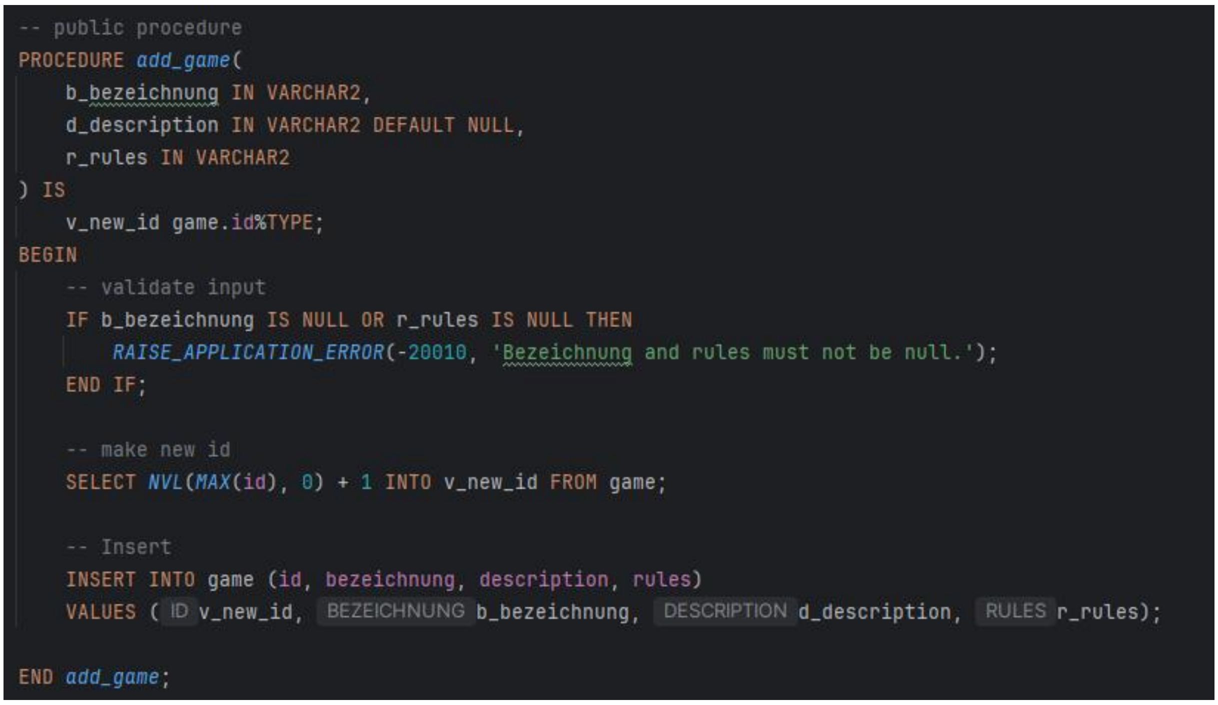


```
-- game package
CREATE OR REPLACE PACKAGE game_api IS
    PROCEDURE add_game(
        b_bezeichnung IN VARCHAR2,
        d_description IN VARCHAR2 DEFAULT NULL,
        r_rules IN VARCHAR2
    );
END game_api;
```



```
CREATE OR REPLACE PACKAGE BODY game_api IS

    -- public procedure
    PROCEDURE add_game(
        b_bezeichnung IN VARCHAR2,
        d_description IN VARCHAR2 DEFAULT NULL,
        r_rules IN VARCHAR2
    ) IS
        v_new_id game.id%TYPE;
    BEGIN
        -- validate input
        IF b_bezeichnung IS NULL OR r_rules IS NULL THEN
            RAISE_APPLICATION_ERROR(-20010, 'Bezeichnung and rules must not be null.');
        END IF;

        -- make new id
        SELECT NVL(MAX(id), 0) + 1 INTO v_new_id FROM game;

        -- Insert
        INSERT INTO game (id, bezeichnung, description, rules)
        VALUES ( ID v_new_id, BEZEICHNUNG b_bezeichnung, DESCRIPTION d_description, RULES r_rules);

    END add_game;

END game_api;
```

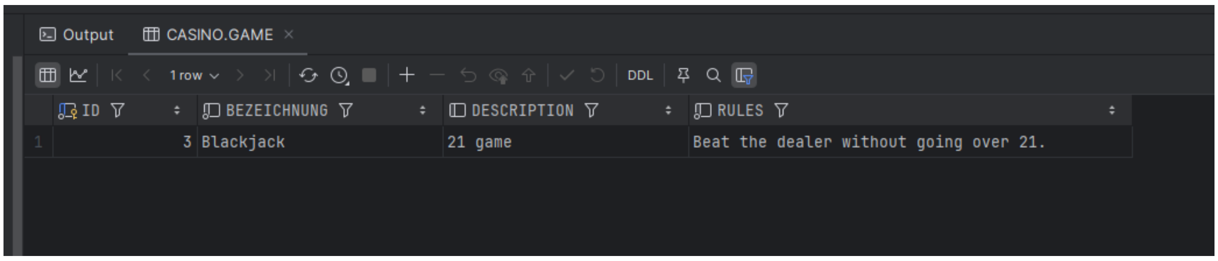
Use Case 5 (Add New Game):



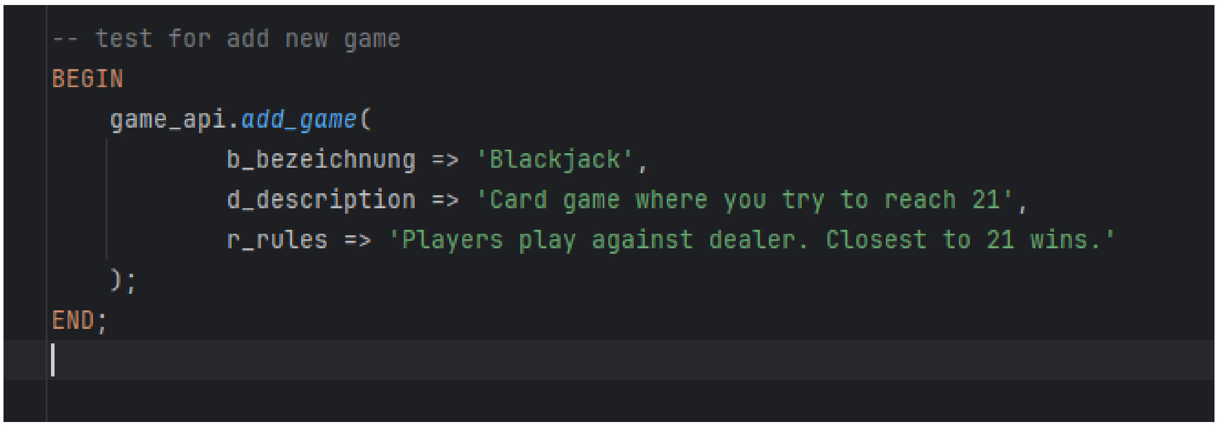
```
-- public procedure
PROCEDURE add_game(
    b_bezeichnung IN VARCHAR2,
    d_description IN VARCHAR2 DEFAULT NULL,
    r_rules IN VARCHAR2
) IS
    v_new_id game.id%TYPE;
BEGIN
    -- validate input
    IF b_bezeichnung IS NULL OR r_rules IS NULL THEN
        RAISE_APPLICATION_ERROR(-20010, 'Bezeichnung and rules must not be null.');
    END IF;

    -- make new id
    SELECT NVL(MAX(id), 0) + 1 INTO v_new_id FROM game;

    -- Insert
    INSERT INTO game (id, bezeichnung, description, rules)
    VALUES ( ID v_new_id,  BEZEICHNUNG b_bezeichnung,  DESCRIPTION d_description,  RULES r_rules);

END add_game;
```

Test for use case 5:

Games with blackjack before:

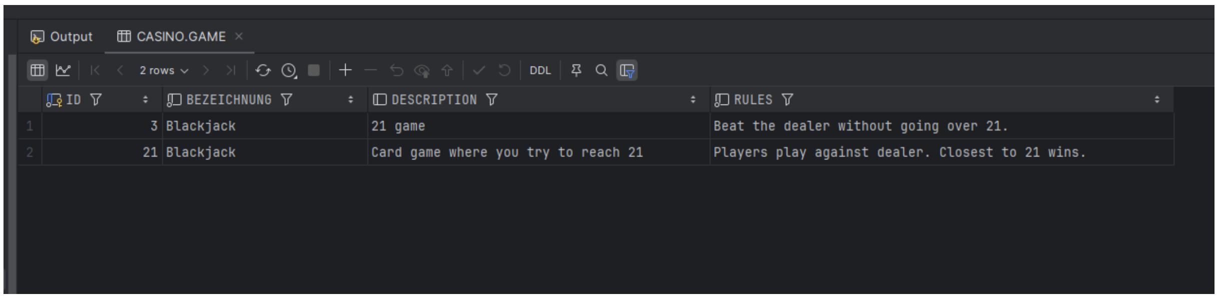


	ID	BEZEICHNUNG	DESCRIPTION	RULES
1	3	Blackjack	21 game	Beat the dealer without going over 21.



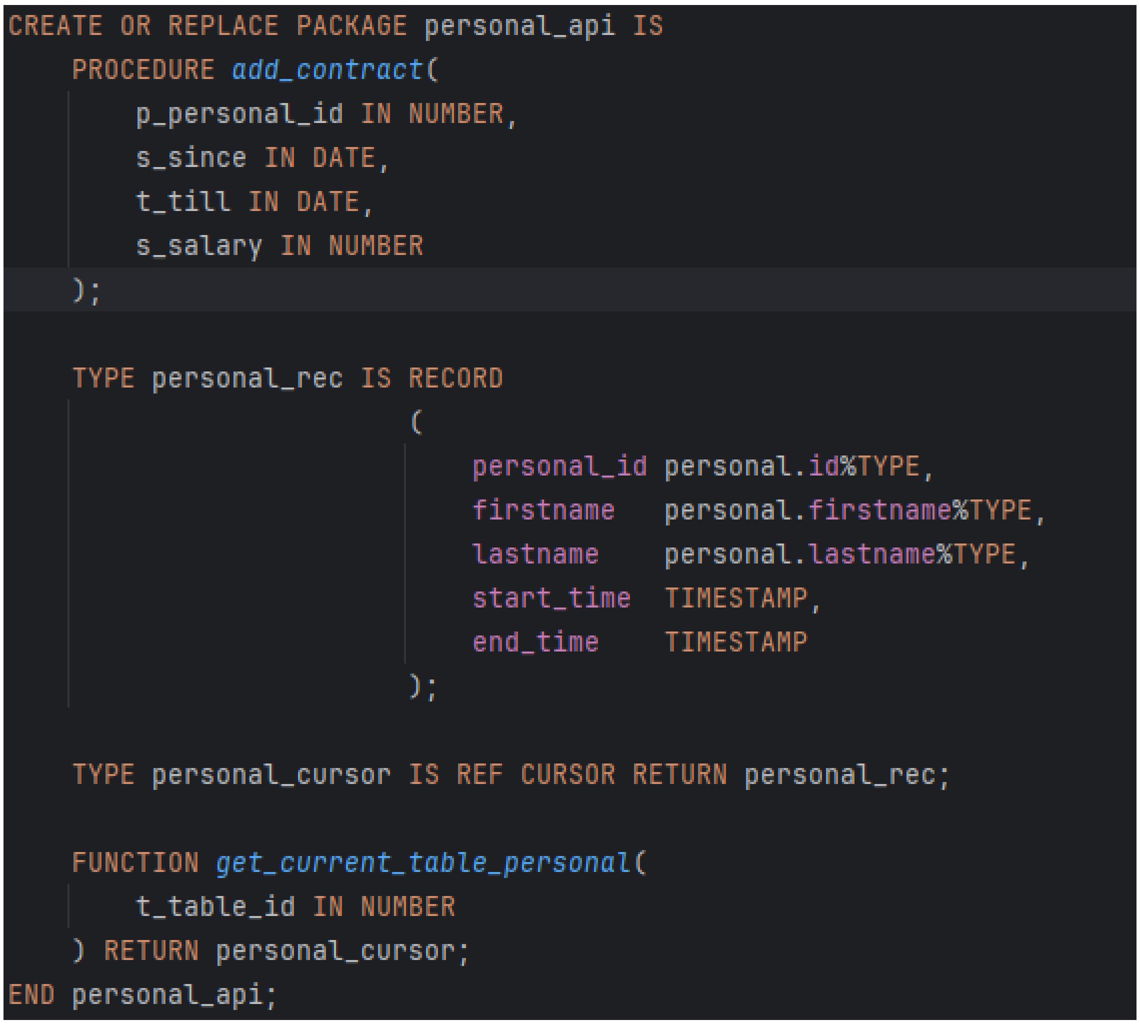
```
-- test for add new game
BEGIN
    game_api.add_game(
            b_bezeichnung => 'Blackjack',
            d_description => 'Card game where you try to reach 21',
            r_rules => 'Players play against dealer. Closest to 21 wins.'
    );
END;
```

After execution:



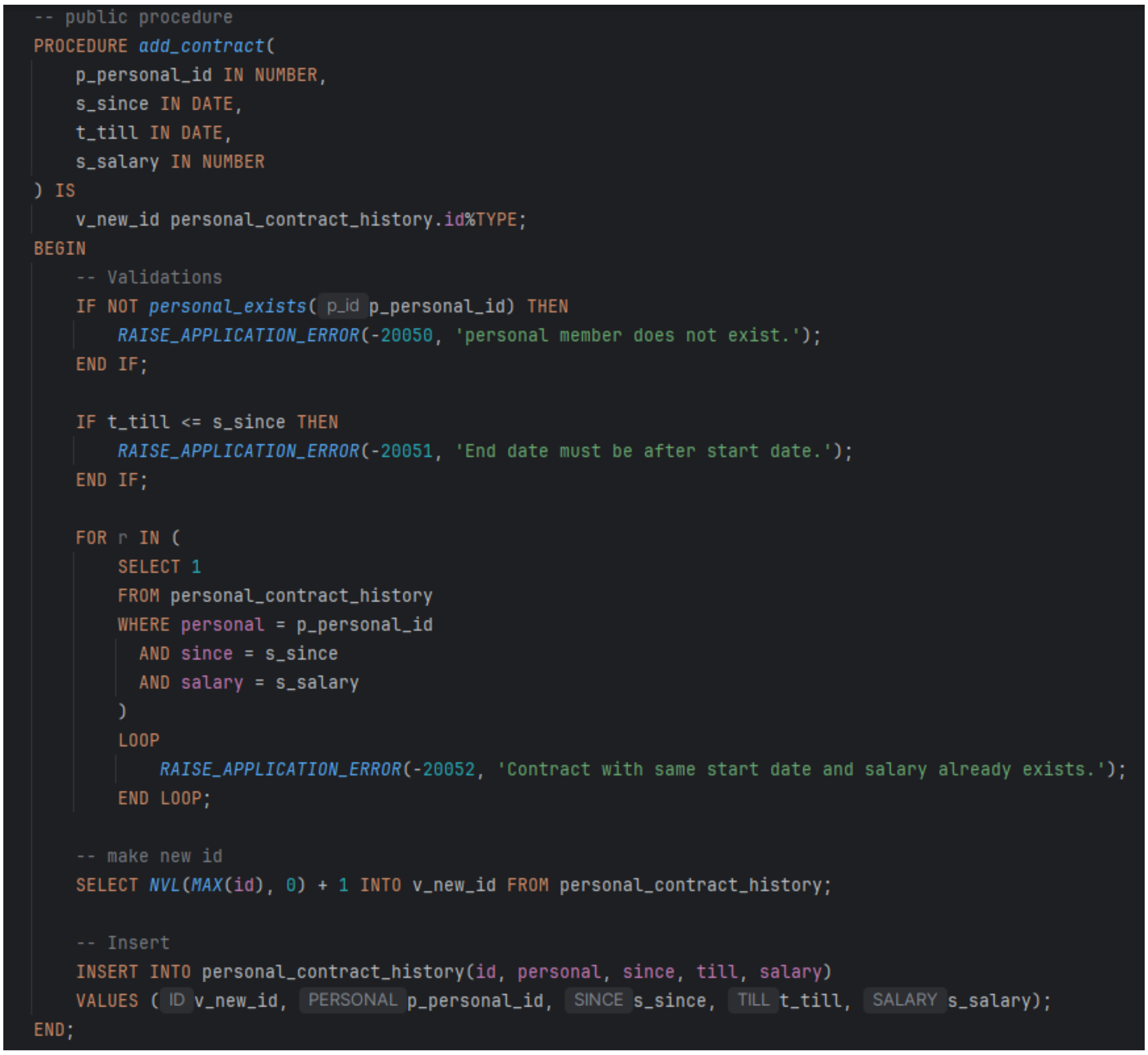
	ID	BEZEICHNUNG	DESCRIPTION	RULES
1	3	Blackjack	21 game	Beat the dealer without going over 21.
2	21	Blackjack	Card game where you try to reach 21	Players play against dealer. Closest to 21 wins.

Personal Use Case 6,7 (add personal contract, get the personal of a table):



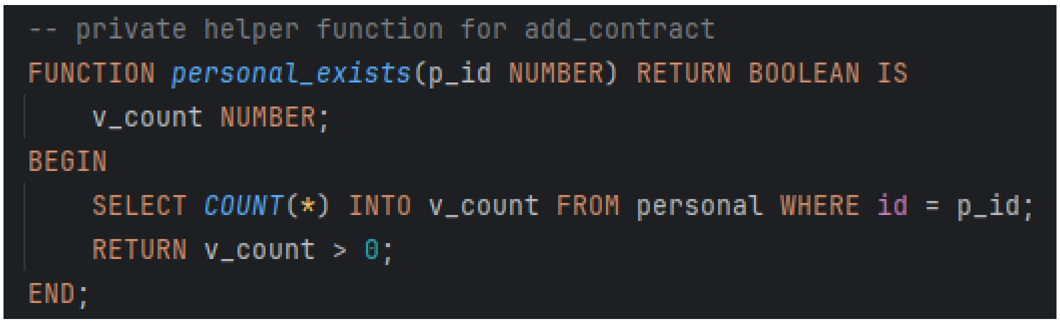
```
CREATE OR REPLACE PACKAGE personal_api IS
    PROCEDURE add_contract(
        p_personal_id IN NUMBER,
        s_since IN DATE,
        t_till IN DATE,
        s_salary IN NUMBER
    );

    TYPE personal_rec IS RECORD
                         (
                             personal_id personal.id%TYPE,
                             firstname   personal.firstname%TYPE,
                             lastname    personal.lastname%TYPE,
                             start_time  TIMESTAMP,
                             end_time    TIMESTAMP
                         );

    TYPE personal_cursor IS REF CURSOR RETURN personal_rec;

    FUNCTION get_current_table_personal(
        t_table_id IN NUMBER
    ) RETURN personal_cursor;
END personal_api;
```

Use case 6 (add personal contract):



```
-- public procedure
PROCEDURE add_contract(
    p_personal_id IN NUMBER,
    s_since IN DATE,
    t_till IN DATE,
    s_salary IN NUMBER
) IS
    v_new_id personal_contract_history.id%TYPE;
BEGIN
    -- Validations
    IF NOT personal_exists( p_id p_personal_id) THEN
        RAISE_APPLICATION_ERROR(-20050, 'personal member does not exist.');
    END IF;

    IF t_till <= s_since THEN
        RAISE_APPLICATION_ERROR(-20051, 'End date must be after start date.');
    END IF;

    FOR r IN (
        SELECT 1
        FROM personal_contract_history
        WHERE personal = p_personal_id
          AND since = s_since
          AND salary = s_salary
        )
        LOOP
            RAISE_APPLICATION_ERROR(-20052, 'Contract with same start date and salary already exists.');
        END LOOP;

    -- make new id
    SELECT NVL(MAX(id), 0) + 1 INTO v_new_id FROM personal_contract_history;

    -- Insert
    INSERT INTO personal_contract_history(id, personal, since, till, salary)
    VALUES ( ID v_new_id, PERSONAL p_personal_id, SINCE s_since, TILL t_till, SALARY s_salary);
END;
```

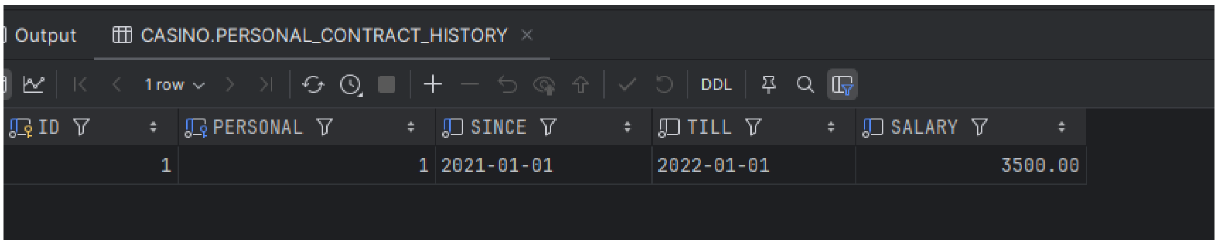
Helper function:



```
-- private helper function for add_contract
FUNCTION personal_exists(p_id NUMBER) RETURN BOOLEAN IS
    v_count NUMBER;
BEGIN
    SELECT COUNT(*) INTO v_count FROM personal WHERE id = p_id;
    RETURN v_count > 0;
END;
```

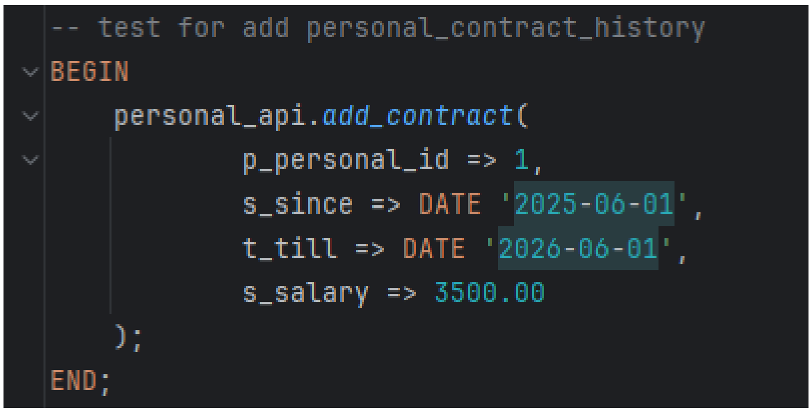
Test for use case 6:

Before execution:



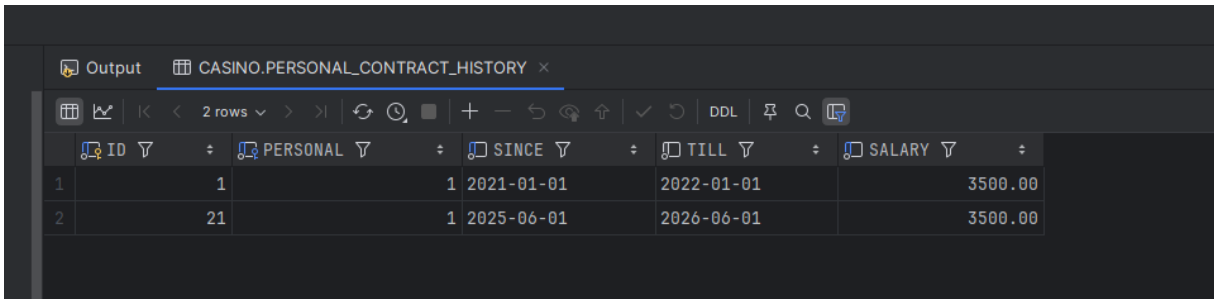
Output | CASINO.PERSONAL_CONTRACT_HISTORY

ID	PERSONAL	SINCE	TILL	SALARY
1	1	2021-01-01	2022-01-01	3500.00



```
-- test for add personal_contract_history
BEGIN
    personal_api.add_contract(
            p_personal_id => 1,
            s_since => DATE '2025-06-01',
            t_till => DATE '2026-06-01',
            s_salary => 3500.00
    );
END;
```

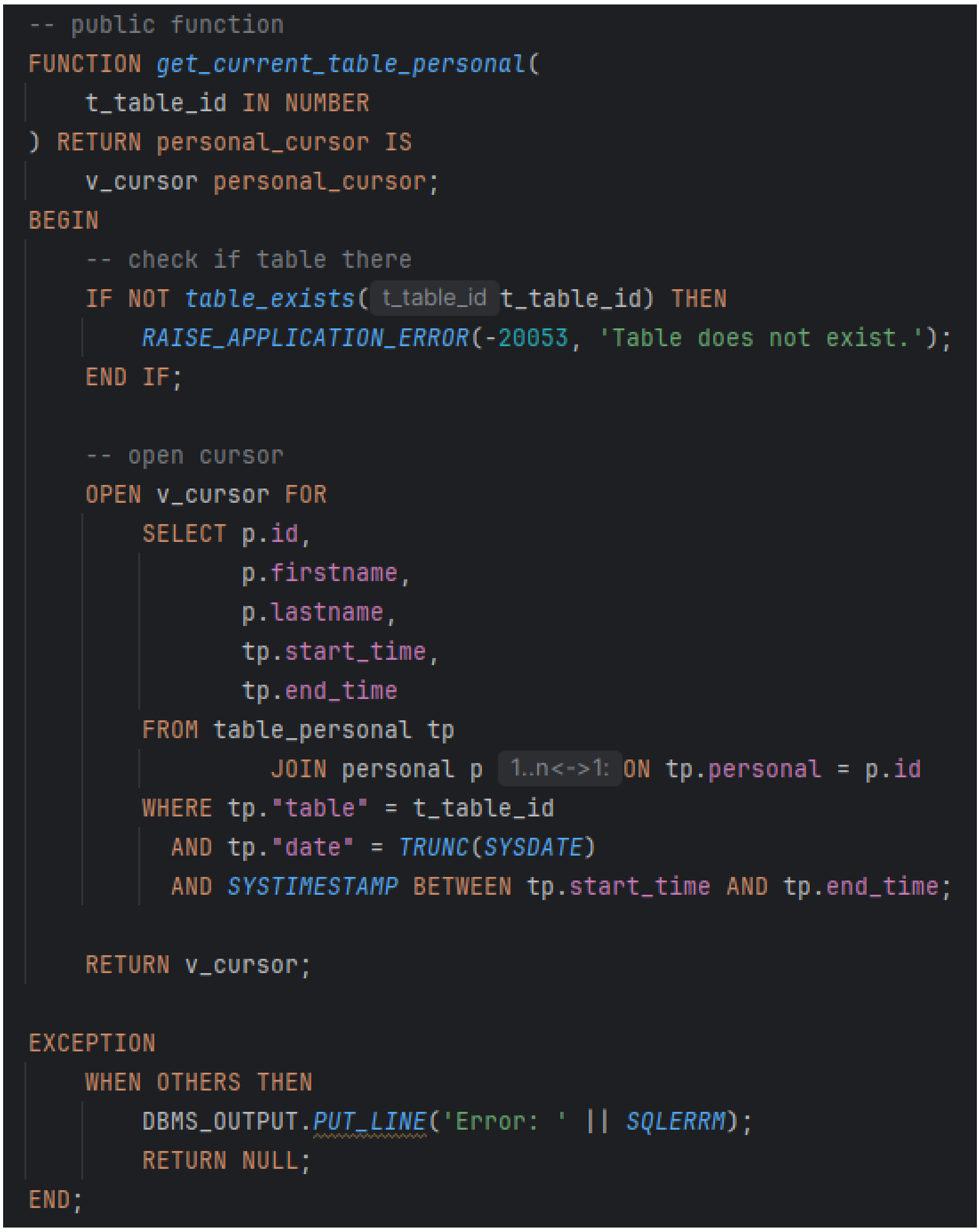
After execution:



Output | CASINO.PERSONAL_CONTRACT_HISTORY

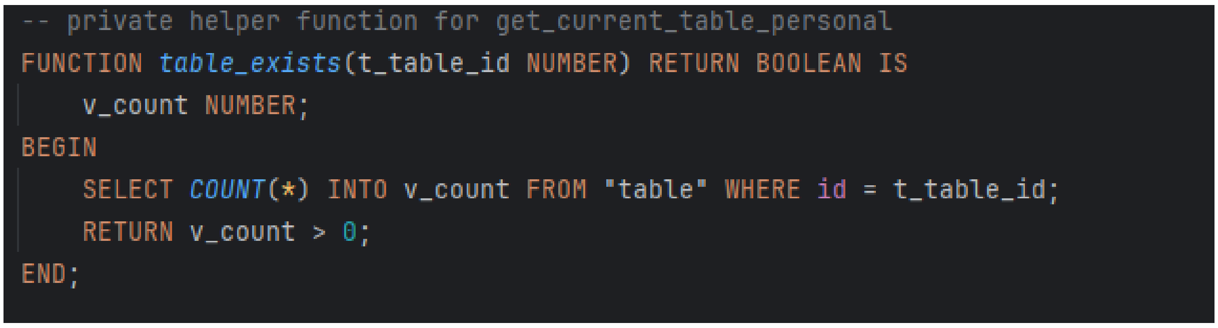
	ID	PERSONAL	SINCE	TILL	SALARY
1	1	1	2021-01-01	2022-01-01	3500.00
2	21	1	2025-06-01	2026-06-01	3500.00

Use Case 7 (get the personal of a table):



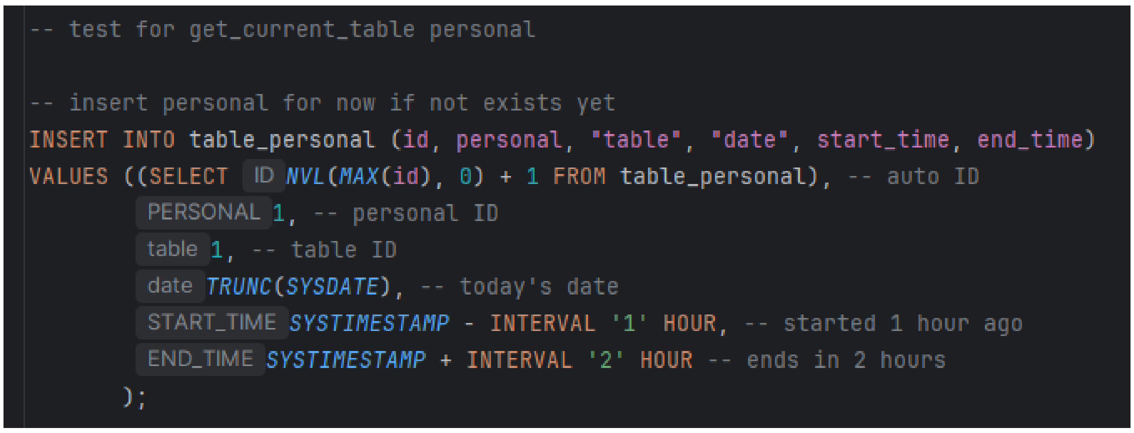
```
-- public function
FUNCTION get_current_table_personal(
    t_table_id IN NUMBER
) RETURN personal_cursor IS
    v_cursor personal_cursor;
BEGIN
    -- check if table there
    IF NOT table_exists( t_table_id t_table_id) THEN
        RAISE_APPLICATION_ERROR(-20053, 'Table does not exist.');
    END IF;

    -- open cursor
    OPEN v_cursor FOR
        SELECT p.id,
               p.firstname,
               p.lastname,
               tp.start_time,
               tp.end_time
        FROM table_personal tp
                 JOIN personal p  1..n<->1: ON tp.personal = p.id
        WHERE tp."table" = t_table_id
          AND tp."date" = TRUNC(SYSDATE)
          AND SYSTIMESTAMP BETWEEN tp.start_time AND tp.end_time;

    RETURN v_cursor;

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

Helper function:

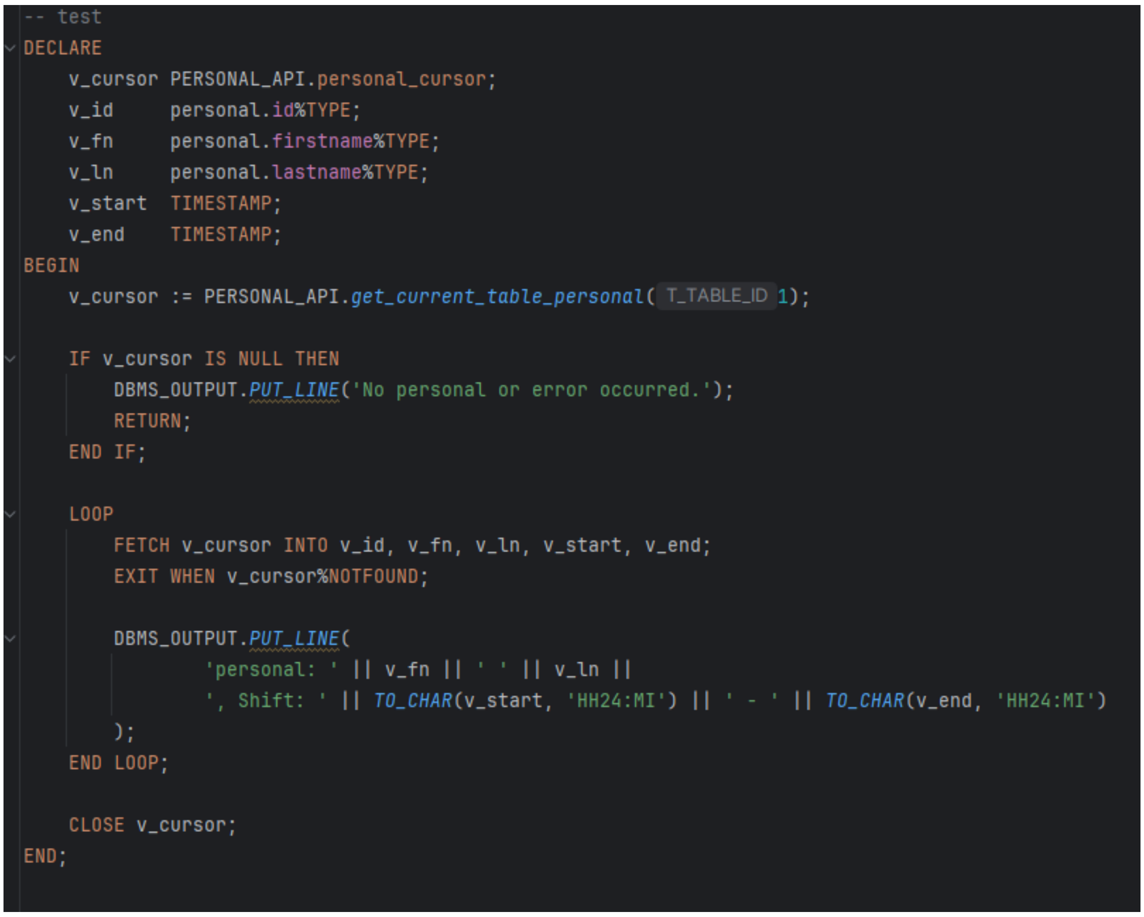


```
-- private helper function for get_current_table_personal
FUNCTION table_exists(t_table_id NUMBER) RETURN BOOLEAN IS
    v_count NUMBER;
BEGIN
    SELECT COUNT(*) INTO v_count FROM "table" WHERE id = t_table_id;
    RETURN v_count > 0;
END;
```

Test for use case 7:

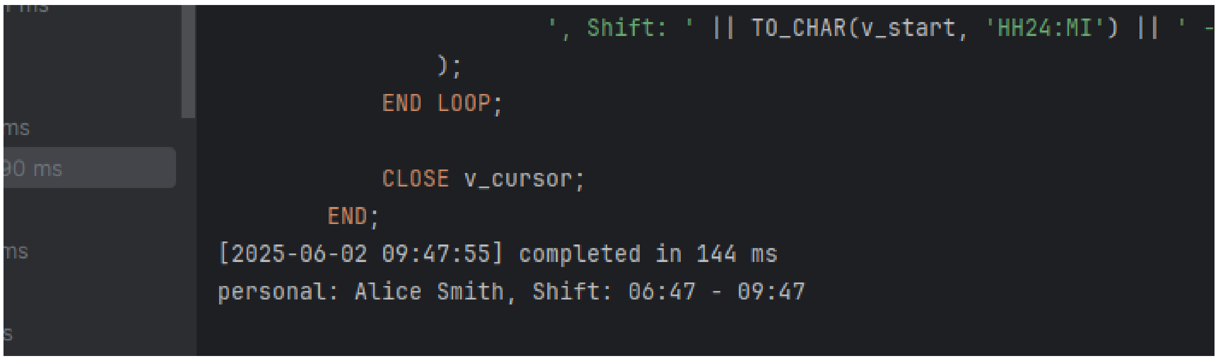


```
-- test for get_current_table personal

-- insert personal for now if not exists yet
INSERT INTO table_personal (id, personal, "table", "date", start_time, end_time)
VALUES ((SELECT  ID NVL(MAX(id), 0) + 1 FROM table_personal), -- auto ID
         PERSONAL 1, -- personal ID
         table 1, -- table ID
         date TRUNC(SYSDATE), -- today's date
         START_TIME SYSTIMESTAMP - INTERVAL '1' HOUR, -- started 1 hour ago
         END_TIME SYSTIMESTAMP + INTERVAL '2' HOUR -- ends in 2 hours
       );
```



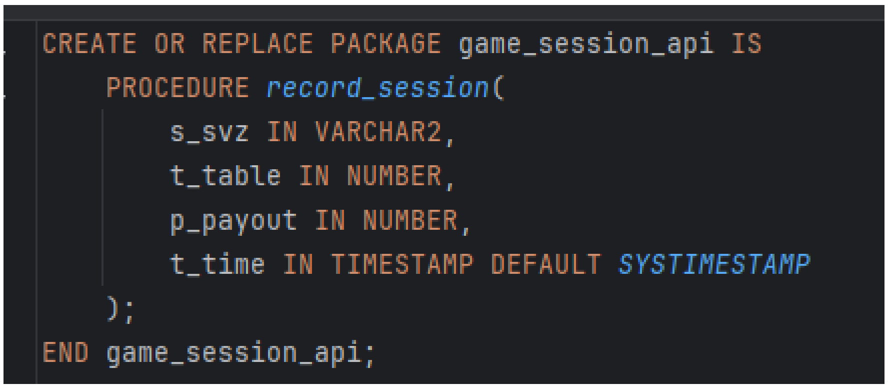
```
-- test
DECLARE
    v_cursor PERSONAL_API.personal_cursor;
    v_id     personal.id%TYPE;
    v_fn     personal.firstname%TYPE;
    v_ln     personal.lastname%TYPE;
    v_start  TIMESTAMP;
    v_end    TIMESTAMP;
BEGIN
    v_cursor := PERSONAL_API.get_current_table_personal( T_TABLE_ID 1);

    IF v_cursor IS NULL THEN
        DBMS_OUTPUT.PUT_LINE('No personal or error occurred.');
        RETURN;
    END IF;

    LOOP
        FETCH v_cursor INTO v_id, v_fn, v_ln, v_start, v_end;
        EXIT WHEN v_cursor%NOTFOUND;

        DBMS_OUTPUT.PUT_LINE(
                'personal: ' || v_fn || ' ' || v_ln ||
                ', Shift: ' || TO_CHAR(v_start, 'HH24:MI') || ' - ' || TO_CHAR(v_end, 'HH24:MI')
        );
    END LOOP;

    CLOSE v_cursor;
END;
```

Prints the personal which is on the table right now and the time:

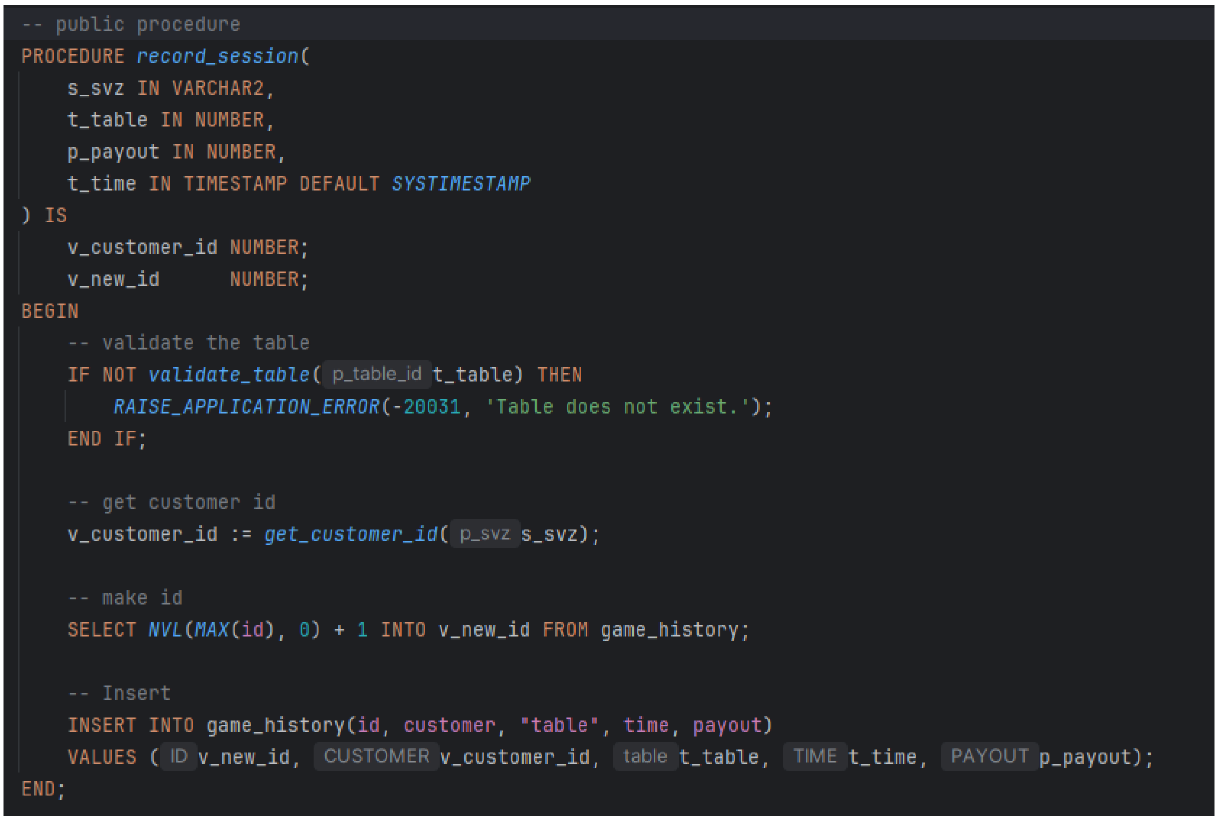


```
                ', Shift: ' || TO_CHAR(v_start, 'HH24:MI') || ' -
        );
    END LOOP;

    CLOSE v_cursor;
END;
[2025-06-02 09:47:55] completed in 144 ms
personal: Alice Smith, Shift: 06:47 - 09:47
```

Use Case 8 (Customer plays a game and wins or loses):

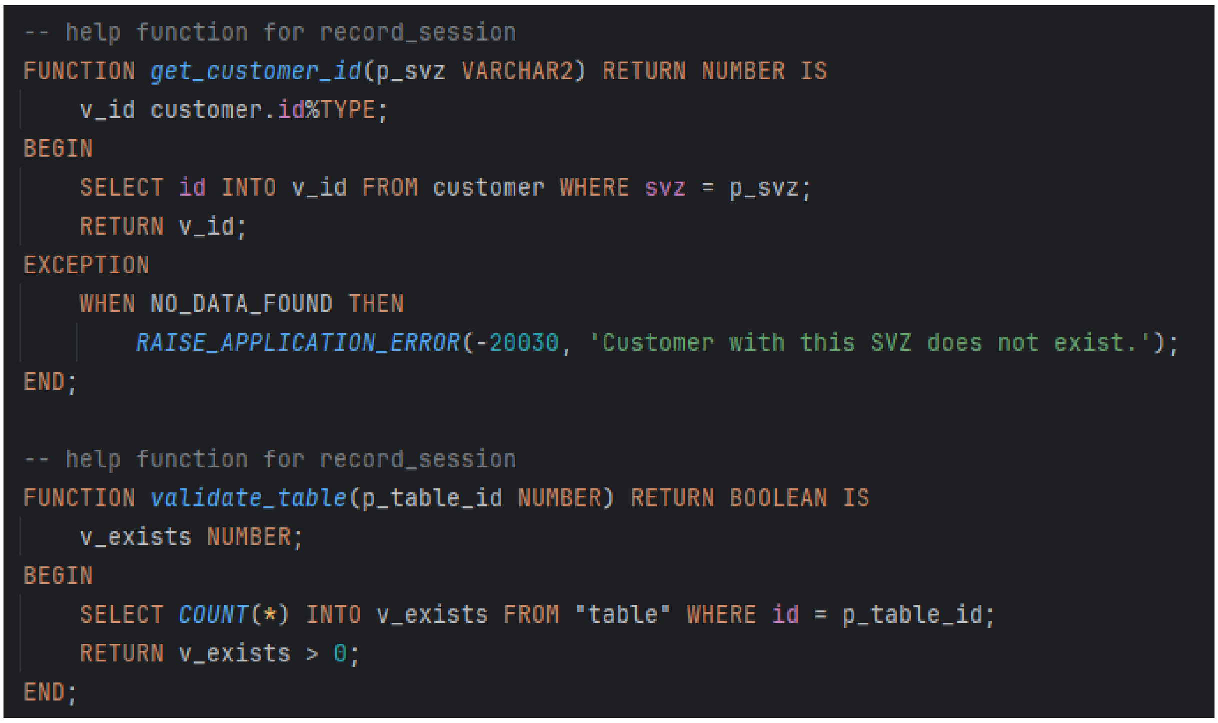


```
CREATE OR REPLACE PACKAGE game_session_api IS
    PROCEDURE record_session(
        s_svz IN VARCHAR2,
        t_table IN NUMBER,
        p_payout IN NUMBER,
        t_time IN TIMESTAMP DEFAULT SYSTIMESTAMP
    );
END game_session_api;
```



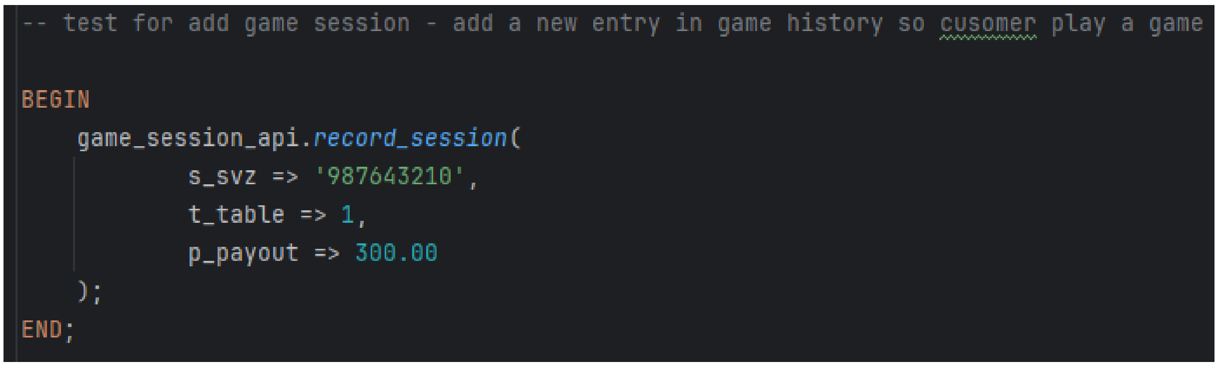
```
-- public procedure
PROCEDURE record_session(
    s_svz IN VARCHAR2,
    t_table IN NUMBER,
    p_payout IN NUMBER,
    t_time IN TIMESTAMP DEFAULT SYSTIMESTAMP
) IS
    v_customer_id NUMBER;
    v_new_id      NUMBER;
BEGIN
    -- validate the table
    IF NOT validate_table( p_table_id t_table) THEN
        RAISE_APPLICATION_ERROR(-20031, 'Table does not exist.');
    END IF;

    -- get customer id
    v_customer_id := get_customer_id( p_svz s_svz);

    -- make id
    SELECT NVL(MAX(id), 0) + 1 INTO v_new_id FROM game_history;

    -- Insert
    INSERT INTO game_history(id, customer, "table", time, payout)
    VALUES ( ID v_new_id,  CUSTOMER v_customer_id,  table t_table,  TIME t_time,  PAYOUT p_payout);
END;
```

Helper functions:



```
-- help function for record_session
FUNCTION get_customer_id(p_svz VARCHAR2) RETURN NUMBER IS
    v_id customer.id%TYPE;
BEGIN
    SELECT id INTO v_id FROM customer WHERE svz = p_svz;
    RETURN v_id;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RAISE_APPLICATION_ERROR(-20030, 'Customer with this SVZ does not exist.');
END;

-- help function for record_session
FUNCTION validate_table(p_table_id NUMBER) RETURN BOOLEAN IS
    v_exists NUMBER;
BEGIN
    SELECT COUNT(*) INTO v_exists FROM "table" WHERE id = p_table_id;
    RETURN v_exists > 0;
END;
```

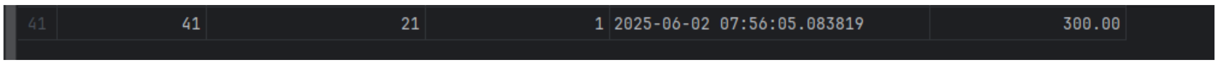
Test for use case 8:



```
-- test for add game session - add a new entry in game history so cusomer play a game

BEGIN
    game_session_api.record_session(
            s_svz => '987643210',
            t_table => 1,
            p_payout => 300.00
    );
END;
```

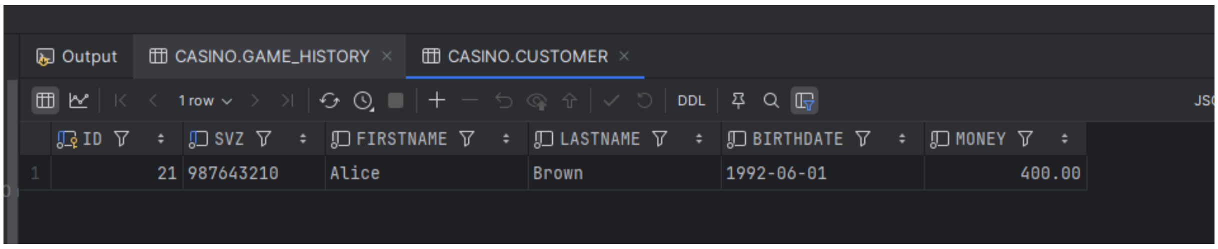
After execution:

Game gets added in game history table:



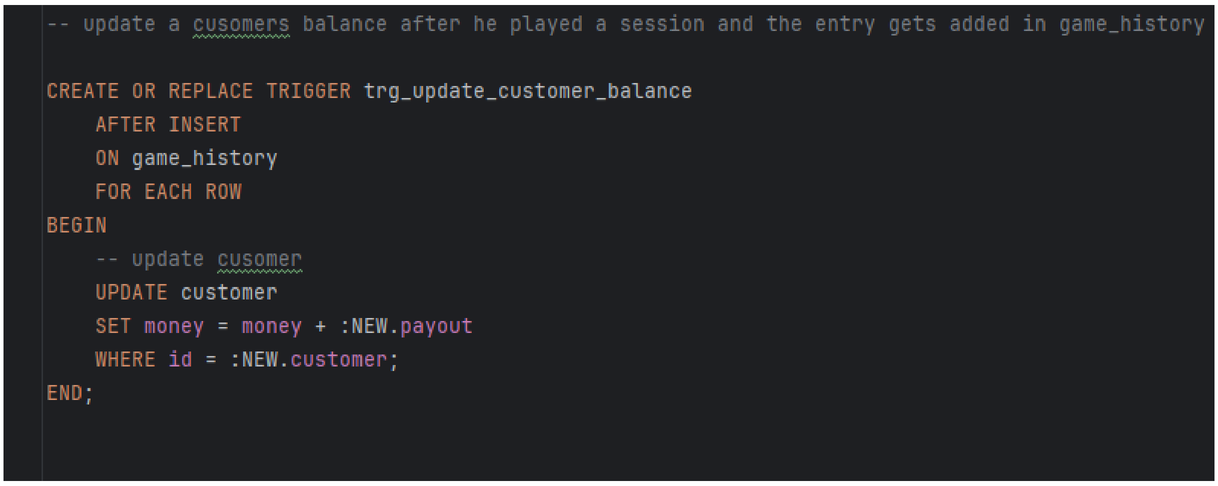
```
41        41        21        1 2025-06-02 07:56:05.083819        300.00
```

+



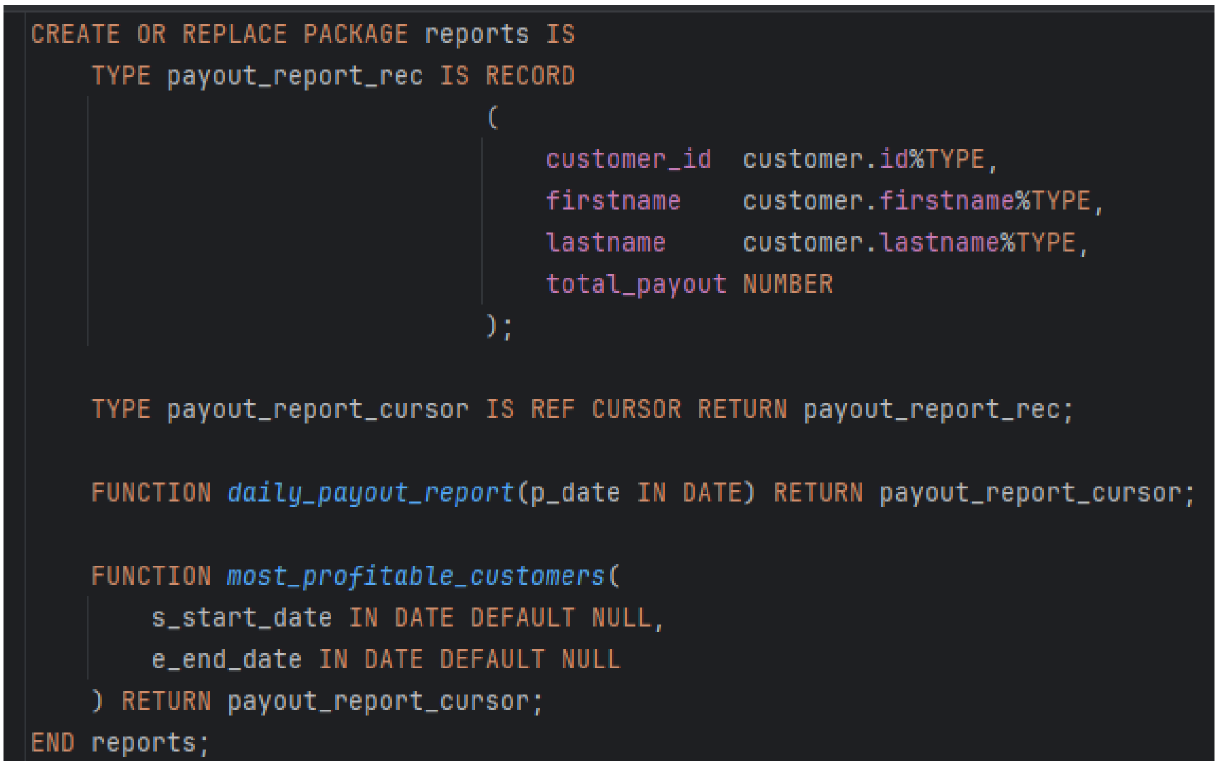
	ID	SVZ	FIRSTNAME	LASTNAME	BIRTHDATE	MONEY
1	21	987643210	Alice	Brown	1992-06-01	400.00

A trigger auto updates the money of the customer:



```
-- update a cusomers balance after he played a session and the entry gets added in game_history

CREATE OR REPLACE TRIGGER trg_update_customer_balance
    AFTER INSERT
    ON game_history
    FOR EACH ROW
BEGIN
    -- update cusomer
    UPDATE customer
    SET money = money + :NEW.payout
    WHERE id = :NEW.customer;
END;
```

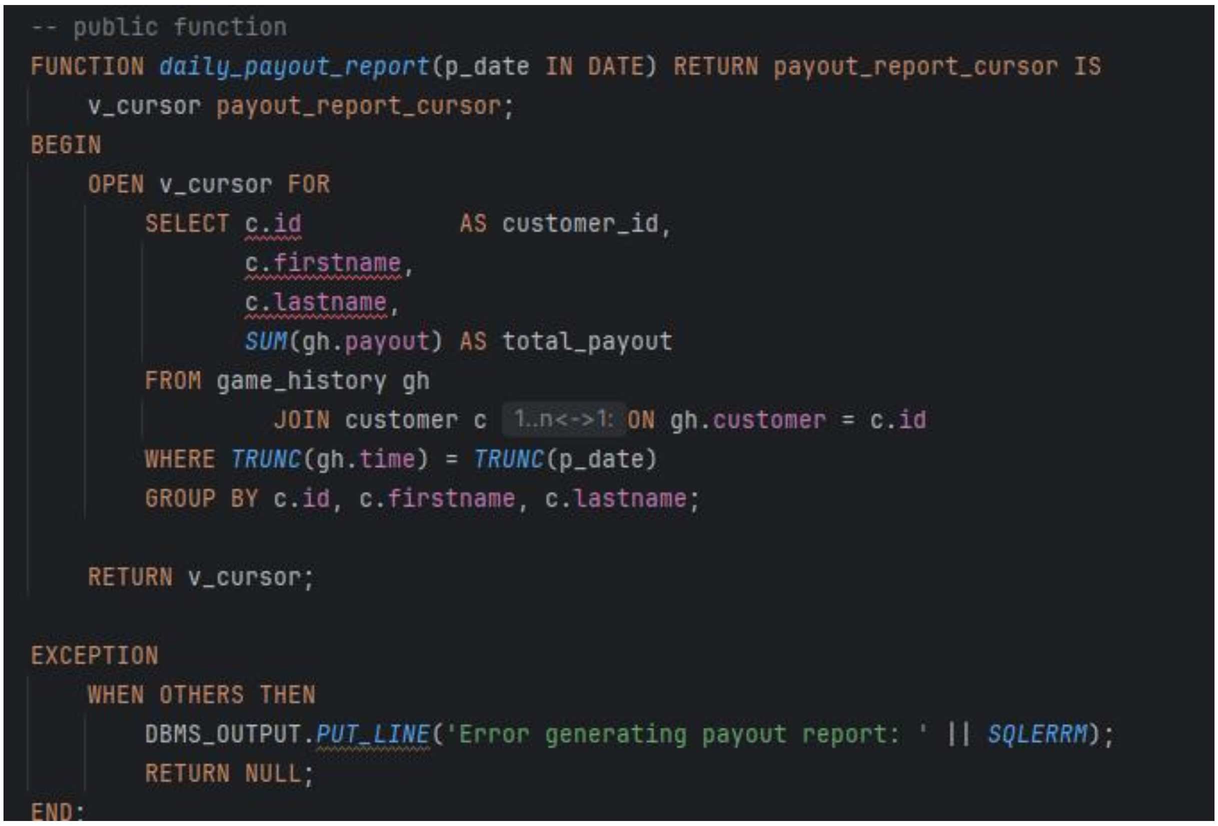
Use Case 9, 10 (make a daily report of the people who played today and the total sum they made, return the money of all customers in descending order):



```
CREATE OR REPLACE PACKAGE reports IS
    TYPE payout_report_rec IS RECORD
                              (
                                  customer_id  customer.id%TYPE,
                                  firstname    customer.firstname%TYPE,
                                  lastname     customer.lastname%TYPE,
                                  total_payout NUMBER
                              );

    TYPE payout_report_cursor IS REF CURSOR RETURN payout_report_rec;

    FUNCTION daily_payout_report(p_date IN DATE) RETURN payout_report_cursor;

    FUNCTION most_profitable_customers(
        s_start_date IN DATE DEFAULT NULL,
        e_end_date IN DATE DEFAULT NULL
    ) RETURN payout_report_cursor;
END reports;
```

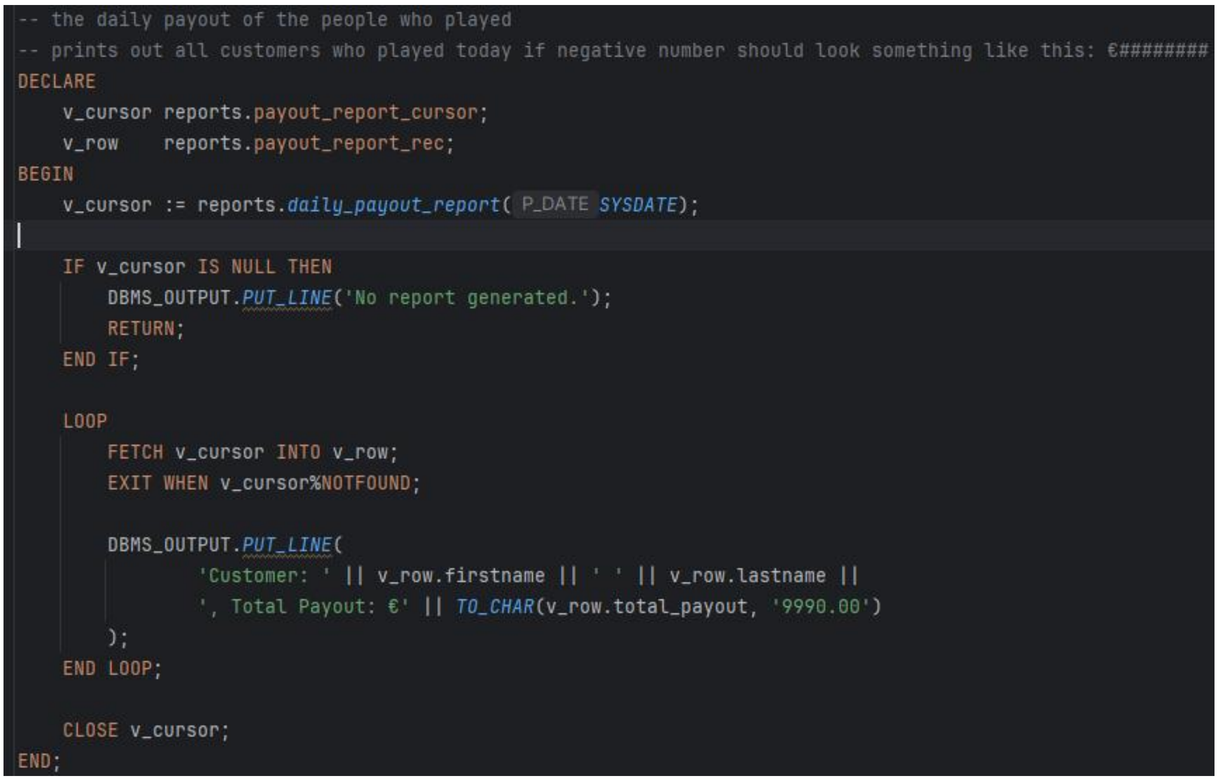
Use Case 9 (make a daily report of the people who played today and the total sum they made):

The sql errors are only shown in the script even though they are not really there



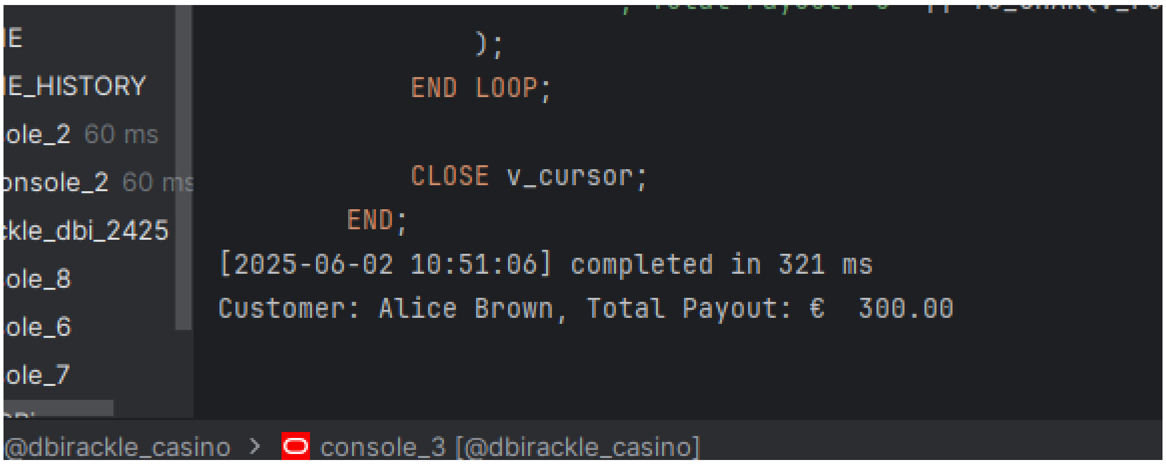
```
-- public function
FUNCTION daily_payout_report(p_date IN DATE) RETURN payout_report_cursor IS
    v_cursor payout_report_cursor;
BEGIN
    OPEN v_cursor FOR
        SELECT c.id           AS customer_id,
               c.firstname,
               c.lastname,
               SUM(gh.payout) AS total_payout
        FROM game_history gh
                 JOIN customer c 1..n<->1: ON gh.customer = c.id
        WHERE TRUNC(gh.time) = TRUNC(p_date)
        GROUP BY c.id, c.firstname, c.lastname;

    RETURN v_cursor;

EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error generating payout report: ' || SQLERRM);
        RETURN NULL;
END;
```

Test for use case 9:



```
-- the daily payout of the people who played
-- prints out all customers who played today if negative number should look something like this: €########
DECLARE
    v_cursor reports.payout_report_cursor;
    v_row    reports.payout_report_rec;
BEGIN
    v_cursor := reports.daily_payout_report( P_DATE SYSDATE);

    IF v_cursor IS NULL THEN
        DBMS_OUTPUT.PUT_LINE('No report generated.');
        RETURN;
    END IF;

    LOOP
        FETCH v_cursor INTO v_row;
        EXIT WHEN v_cursor%NOTFOUND;

        DBMS_OUTPUT.PUT_LINE(
                'Customer: ' || v_row.firstname || ' ' || v_row.lastname ||
                ', Total Payout: €' || TO_CHAR(v_row.total_payout, '9990.00')
        );
    END LOOP;

    CLOSE v_cursor;
END;
```

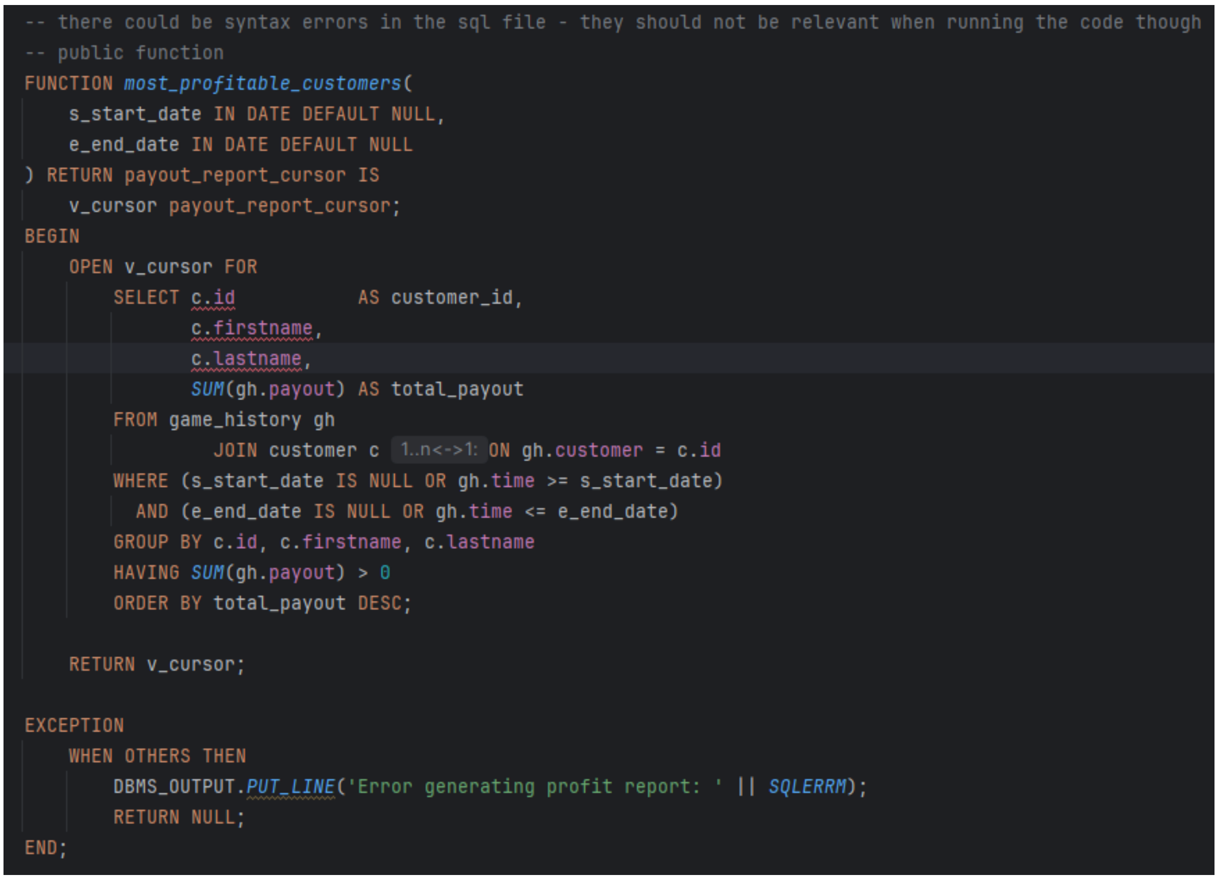
Alle ergebnisse werden ausgeprinted



```
            );
        END LOOP;

        CLOSE v_cursor;
    END;
[2025-06-02 10:51:06] completed in 321 ms
Customer: Alice Brown, Total Payout: €  300.00
```

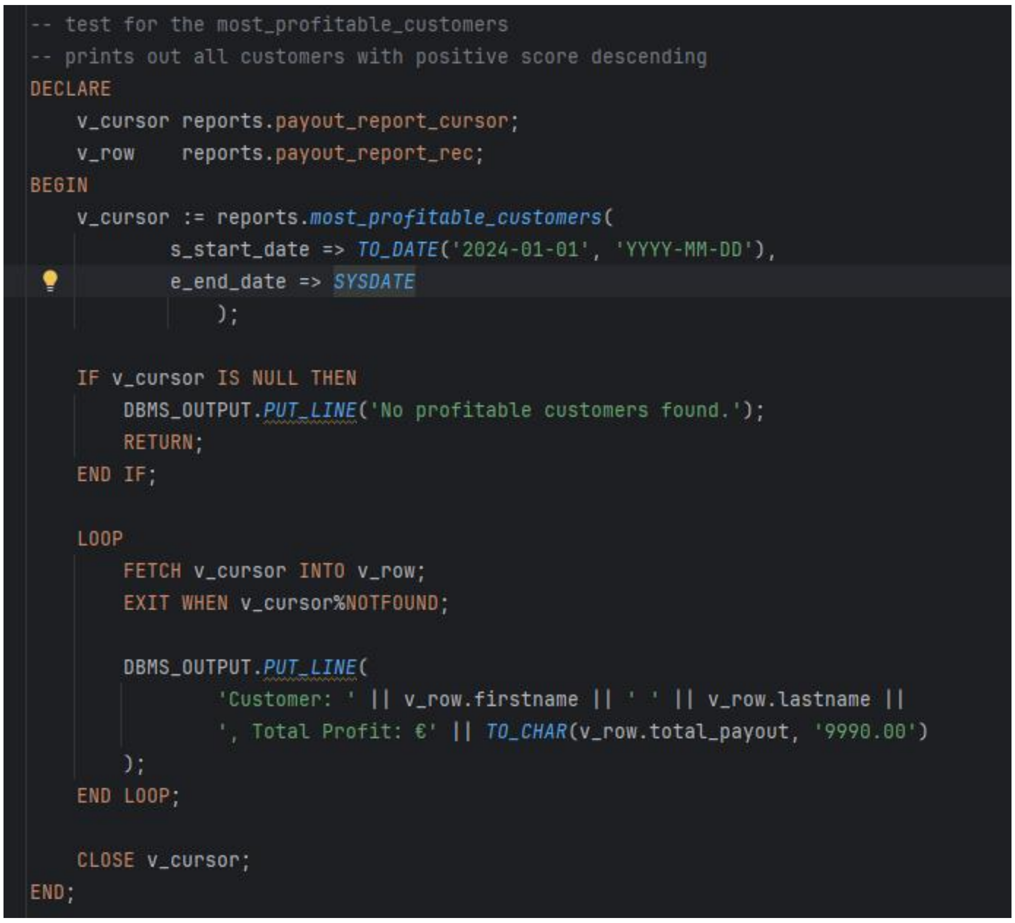
Dieser customer hat heute nur gespielt und insgesamt 300€ dazugewonnen

Use Case 10 (return the money of all customers in descending order):

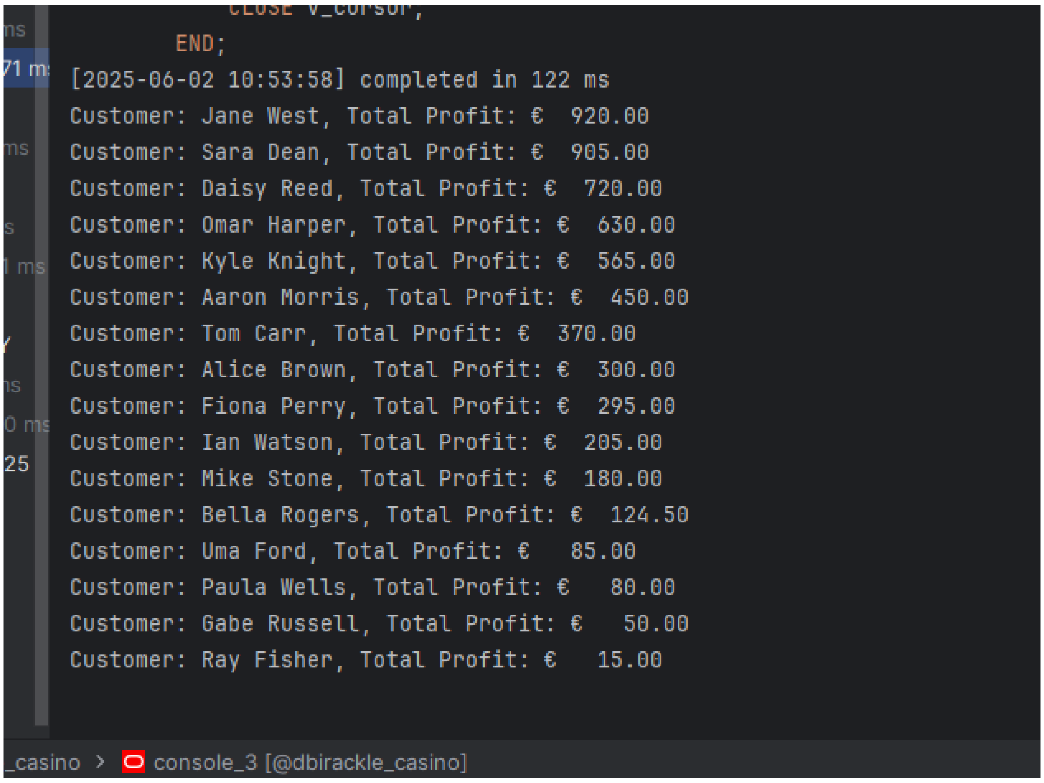


```
-- there could be syntax errors in the sql file - they should not be relevant when running the code though
-- public function
FUNCTION most_profitable_customers(
    s_start_date IN DATE DEFAULT NULL,
    e_end_date IN DATE DEFAULT NULL
) RETURN payout_report_cursor IS
    v_cursor payout_report_cursor;
BEGIN
    OPEN v_cursor FOR
        SELECT c.id           AS customer_id,
               c.firstname,
               c.lastname,
               SUM(gh.payout) AS total_payout
        FROM game_history gh
                 JOIN customer c 1..n<->1: ON gh.customer = c.id
        WHERE (s_start_date IS NULL OR gh.time >= s_start_date)
          AND (e_end_date IS NULL OR gh.time <= e_end_date)
        GROUP BY c.id, c.firstname, c.lastname
        HAVING SUM(gh.payout) > 0
        ORDER BY total_payout DESC;

    RETURN v_cursor;

EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error generating profit report: ' || SQLERRM);
        RETURN NULL;
END;
```

Test for use Case 10:



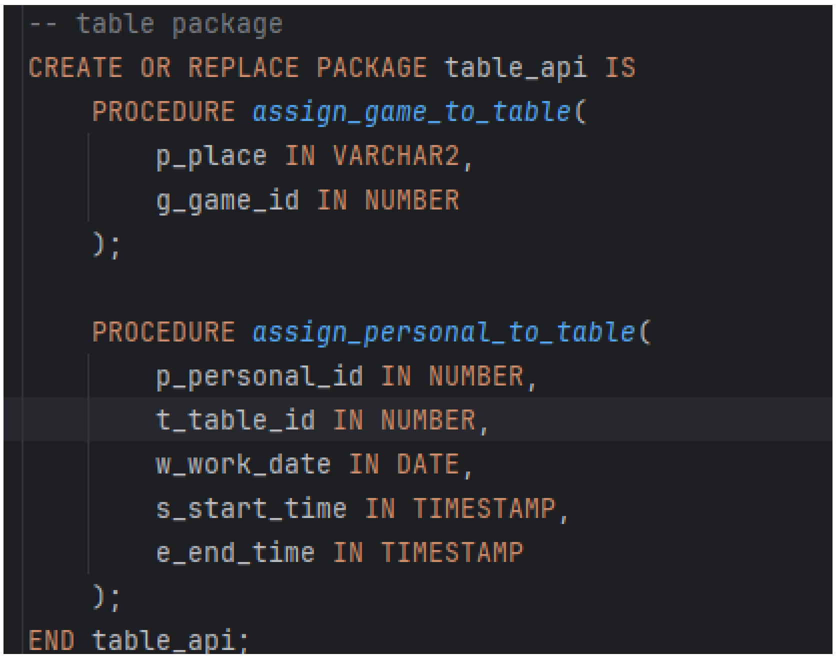
```
-- test for the most_profitable_customers
-- prints out all customers with positive score descending
DECLARE
    v_cursor reports.payout_report_cursor;
    v_row    reports.payout_report_rec;
BEGIN
    v_cursor := reports.most_profitable_customers(
            s_start_date => TO_DATE('2024-01-01', 'YYYY-MM-DD'),
            e_end_date => SYSDATE
                );

    IF v_cursor IS NULL THEN
        DBMS_OUTPUT.PUT_LINE('No profitable customers found.');
        RETURN;
    END IF;

    LOOP
        FETCH v_cursor INTO v_row;
        EXIT WHEN v_cursor%NOTFOUND;

        DBMS_OUTPUT.PUT_LINE(
                'Customer: ' || v_row.firstname || ' ' || v_row.lastname ||
                ', Total Profit: €' || TO_CHAR(v_row.total_payout, '9990.00')
        );
    END LOOP;

    CLOSE v_cursor;
END;
```



```
          END;
[2025-06-02 10:53:58] completed in 122 ms
Customer: Jane West, Total Profit: €  920.00
Customer: Sara Dean, Total Profit: €  905.00
Customer: Daisy Reed, Total Profit: €  720.00
Customer: Omar Harper, Total Profit: €  630.00
Customer: Kyle Knight, Total Profit: €  565.00
Customer: Aaron Morris, Total Profit: €  450.00
Customer: Tom Carr, Total Profit: €  370.00
Customer: Alice Brown, Total Profit: €  300.00
Customer: Fiona Perry, Total Profit: €  295.00
Customer: Ian Watson, Total Profit: €  205.00
Customer: Mike Stone, Total Profit: €  180.00
Customer: Bella Rogers, Total Profit: €  124.50
Customer: Uma Ford, Total Profit: €   85.00
Customer: Paula Wells, Total Profit: €   80.00
Customer: Gabe Russell, Total Profit: €   50.00
Customer: Ray Fisher, Total Profit: €   15.00
```

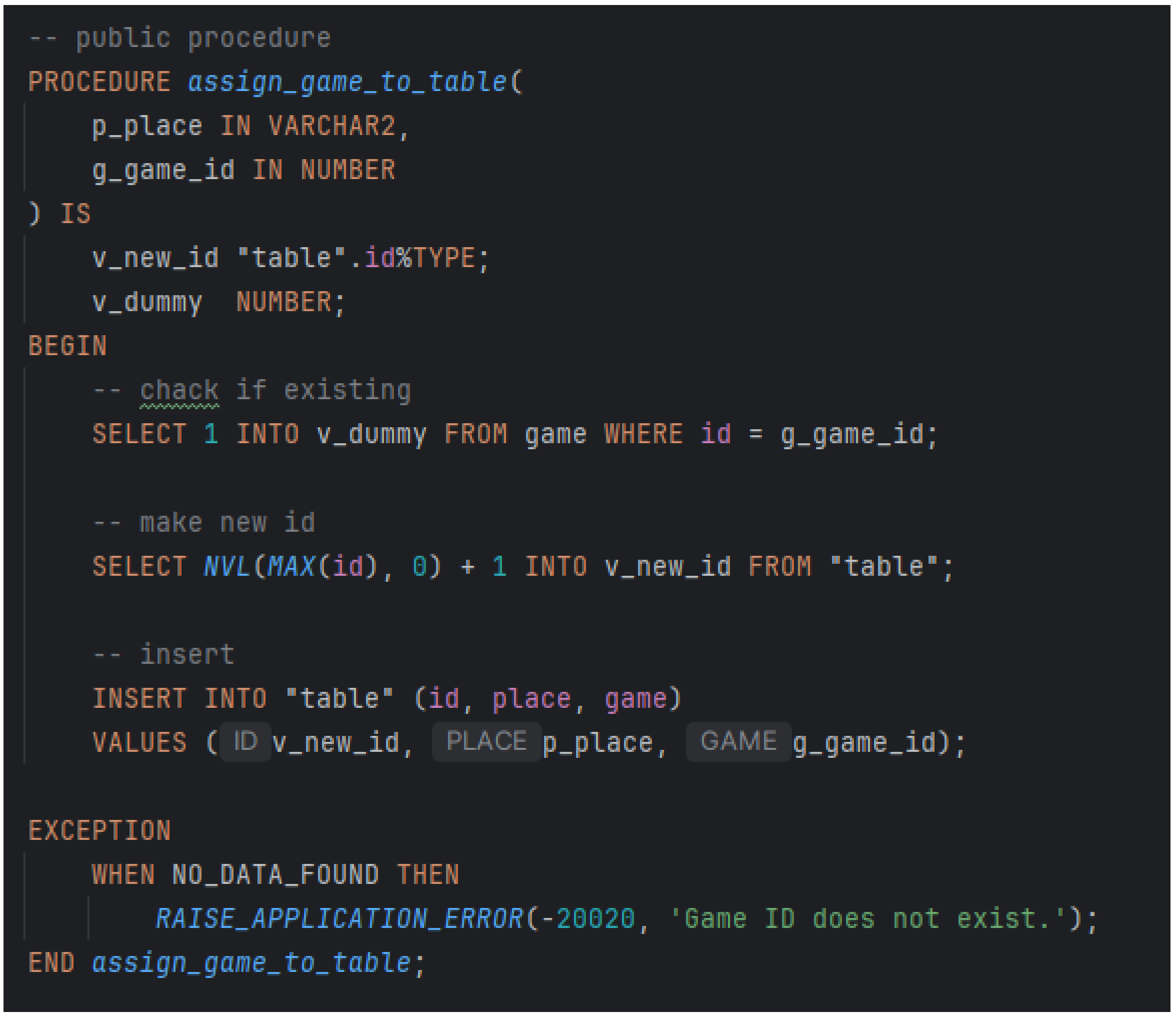
Alle customers mit positive payout werden ausgegeben.

Use Case 11,12 (assign a game to a table, assign personal to a table for a time):



```
-- table package
CREATE OR REPLACE PACKAGE table_api IS
    PROCEDURE assign_game_to_table(
        p_place IN VARCHAR2,
        g_game_id IN NUMBER
    );

    PROCEDURE assign_personal_to_table(
        p_personal_id IN NUMBER,
        t_table_id IN NUMBER,
        w_work_date IN DATE,
        s_start_time IN TIMESTAMP,
        e_end_time IN TIMESTAMP
    );
END table_api;
```

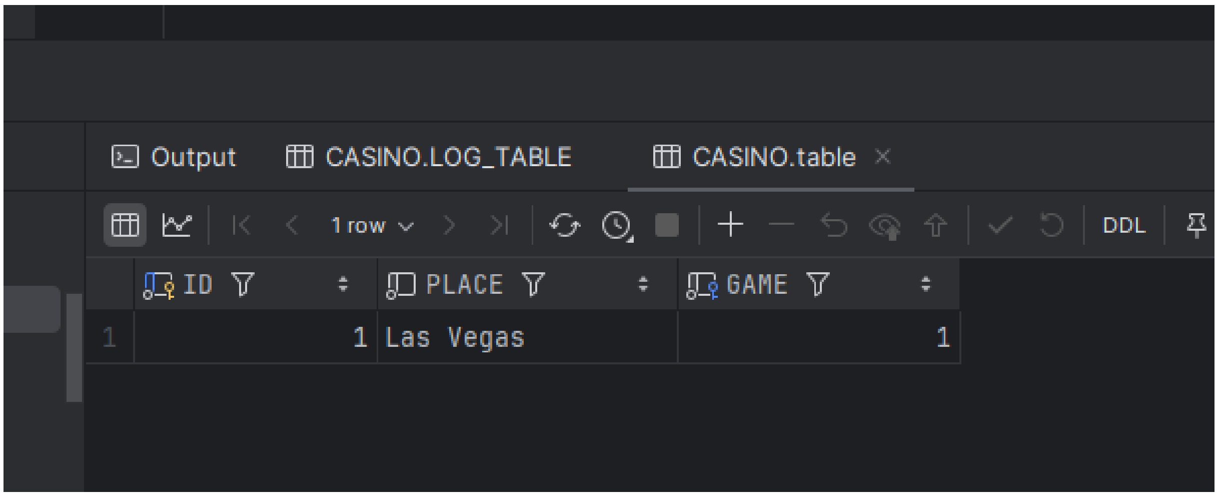
Use Case 11 (assign a game to a table):



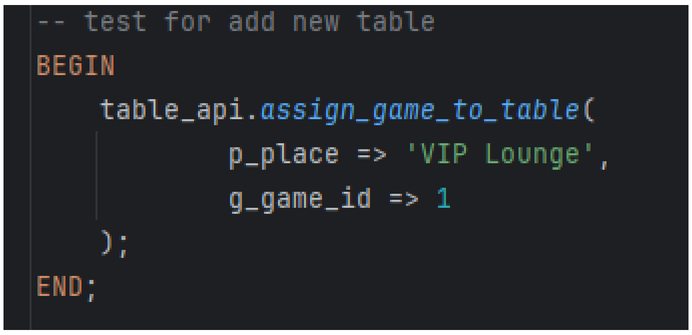
```
-- public procedure
PROCEDURE assign_game_to_table(
    p_place IN VARCHAR2,
    g_game_id IN NUMBER
) IS
    v_new_id "table".id%TYPE;
    v_dummy  NUMBER;
BEGIN
    -- chack if existing
    SELECT 1 INTO v_dummy FROM game WHERE id = g_game_id;

    -- make new id
    SELECT NVL(MAX(id), 0) + 1 INTO v_new_id FROM "table";

    -- insert
    INSERT INTO "table" (id, place, game)
    VALUES ( ID v_new_id, PLACE p_place, GAME g_game_id);

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RAISE_APPLICATION_ERROR(-20020, 'Game ID does not exist.');
END assign_game_to_table;
```

Test for use case 11:

All bevore:

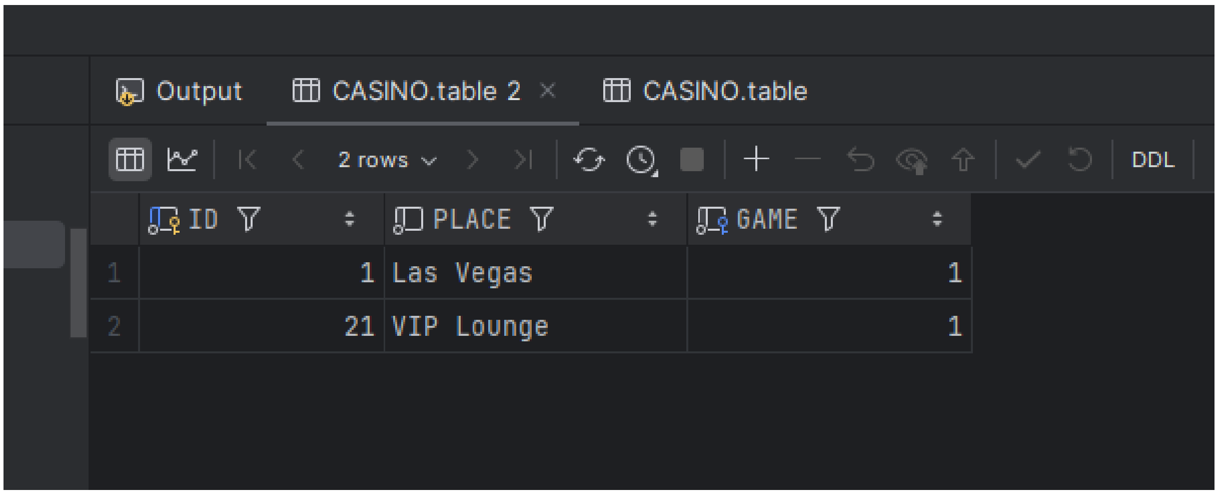


	ID	PLACE	GAME
1	1	Las Vegas	1



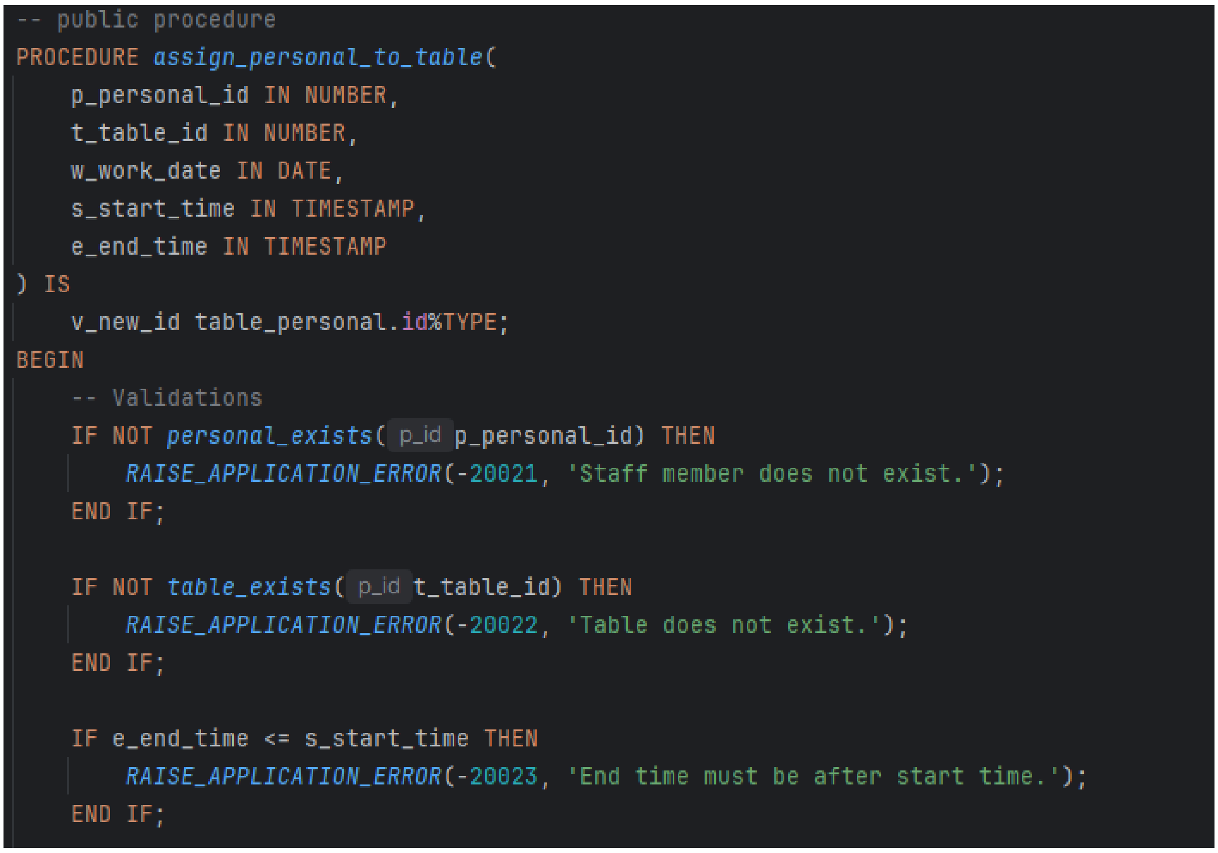
```
-- test for add new table
BEGIN
    table_api.assign_game_to_table(
            p_place => 'VIP Lounge',
            g_game_id => 1
    );
END;
```

After execution:

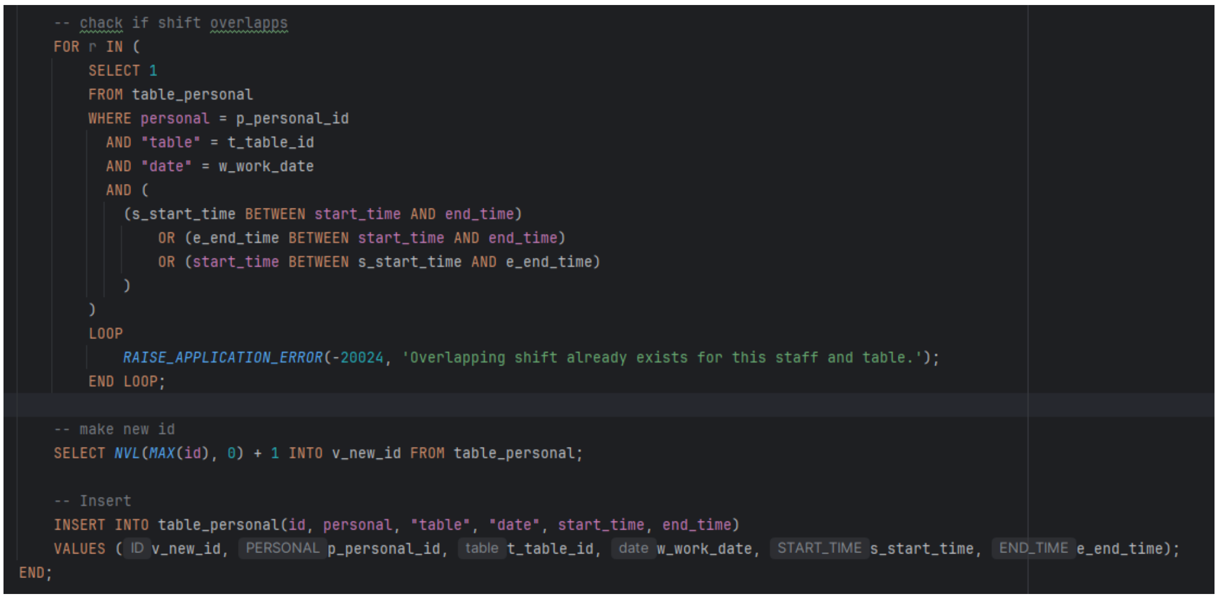


	ID	PLACE	GAME
1	1	Las Vegas	1
2	21	VIP Lounge	1

Use Case 12 (assign personal to a table for a time):

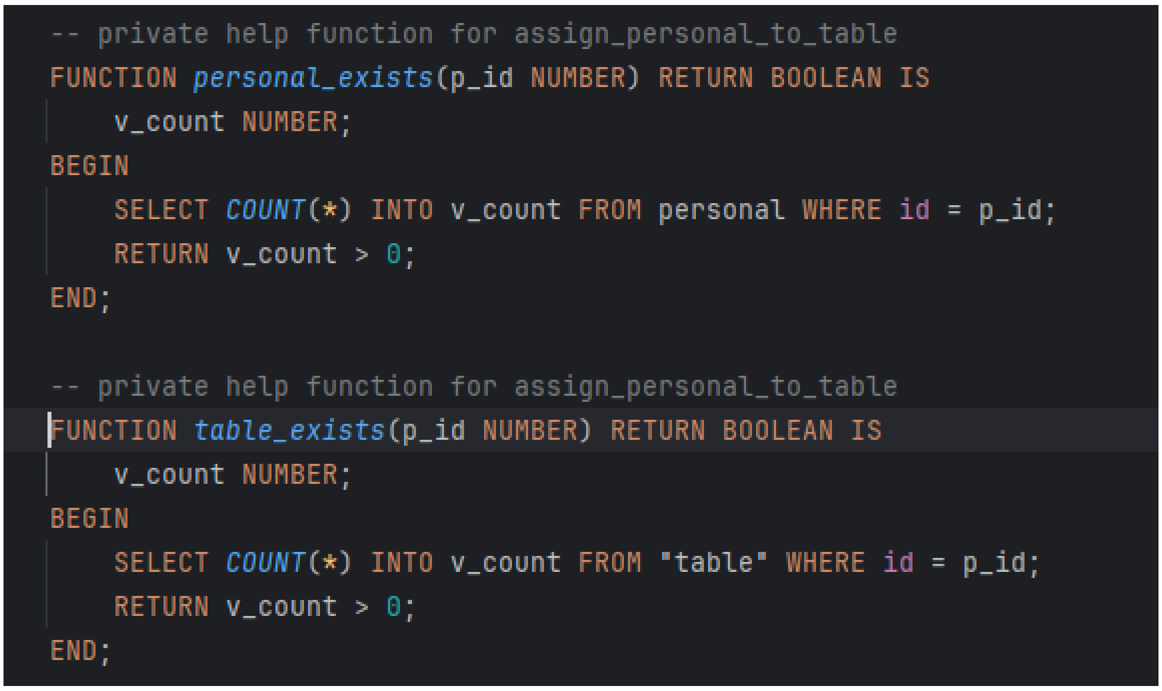


```
-- public procedure
PROCEDURE assign_personal_to_table(
    p_personal_id IN NUMBER,
    t_table_id IN NUMBER,
    w_work_date IN DATE,
    s_start_time IN TIMESTAMP,
    e_end_time IN TIMESTAMP
) IS
    v_new_id table_personal.id%TYPE;
BEGIN
    -- Validations
    IF NOT personal_exists( p_id p_personal_id) THEN
        RAISE_APPLICATION_ERROR(-20021, 'Staff member does not exist.');
    END IF;

    IF NOT table_exists( p_id t_table_id) THEN
        RAISE_APPLICATION_ERROR(-20022, 'Table does not exist.');
    END IF;

    IF e_end_time <= s_start_time THEN
        RAISE_APPLICATION_ERROR(-20023, 'End time must be after start time.');
    END IF;
```



```
    -- chack if shift overlapps
    FOR r IN (
        SELECT 1
        FROM table_personal
        WHERE personal = p_personal_id
          AND "table" = t_table_id
          AND "date" = w_work_date
          AND (
            (s_start_time BETWEEN start_time AND end_time)
                OR (e_end_time BETWEEN start_time AND end_time)
                OR (start_time BETWEEN s_start_time AND e_end_time)
            )
        )
        LOOP
            RAISE_APPLICATION_ERROR(-20024, 'Overlapping shift already exists for this staff and table.');
        END LOOP;

    -- make new id
    SELECT NVL(MAX(id), 0) + 1 INTO v_new_id FROM table_personal;

    -- Insert
    INSERT INTO table_personal(id, personal, "table", "date", start_time, end_time)
    VALUES ( ID v_new_id,  PERSONAL p_personal_id,  table t_table_id,  date w_work_date,  START_TIME s_start_time,  END_TIME e_end_time);
END;
```

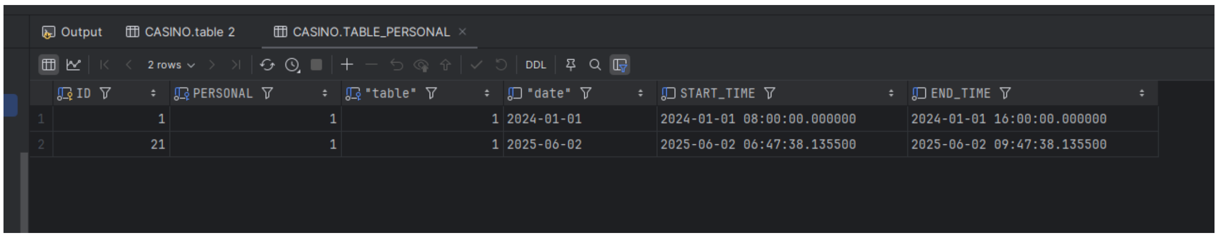
Helper functions:



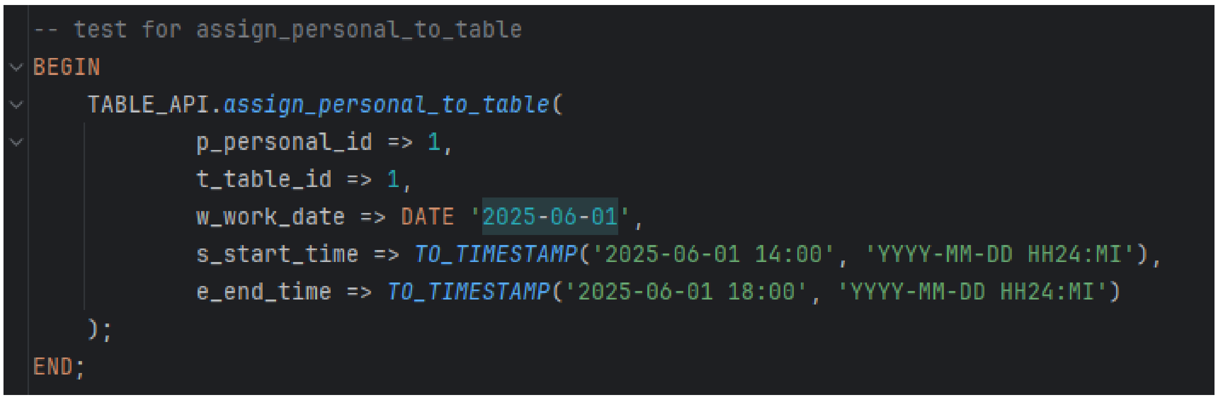
```
-- private help function for assign_personal_to_table
FUNCTION personal_exists(p_id NUMBER) RETURN BOOLEAN IS
    v_count NUMBER;
BEGIN
    SELECT COUNT(*) INTO v_count FROM personal WHERE id = p_id;
    RETURN v_count > 0;
END;

-- private help function for assign_personal_to_table
FUNCTION table_exists(p_id NUMBER) RETURN BOOLEAN IS
    v_count NUMBER;
BEGIN
    SELECT COUNT(*) INTO v_count FROM "table" WHERE id = p_id;
    RETURN v_count > 0;
END;
```

Test for use case 12:

Before execution:

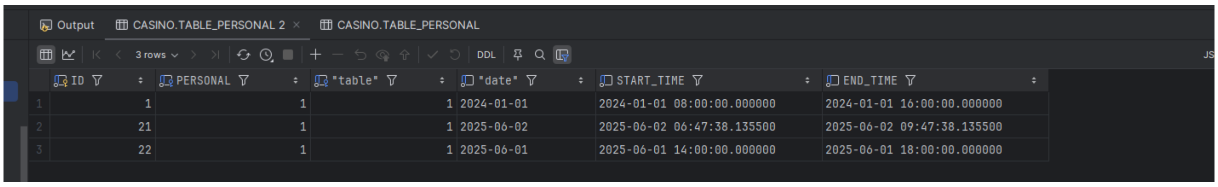


ID	PERSONAL	"table"	"date"	START_TIME	END_TIME
1	1	1	2024-01-01	2024-01-01 08:00:00.000000	2024-01-01 16:00:00.000000
21	1	1	2025-06-02	2025-06-02 06:47:38.135500	2025-06-02 09:47:38.135500



```sql
-- test for assign_personal_to_table
BEGIN
    TABLE_API.assign_personal_to_table(
            p_personal_id => 1,
            t_table_id => 1,
            w_work_date => DATE '2025-06-01',
            s_start_time => TO_TIMESTAMP('2025-06-01 14:00', 'YYYY-MM-DD HH24:MI'),
            e_end_time => TO_TIMESTAMP('2025-06-01 18:00', 'YYYY-MM-DD HH24:MI')
    );
END;
```

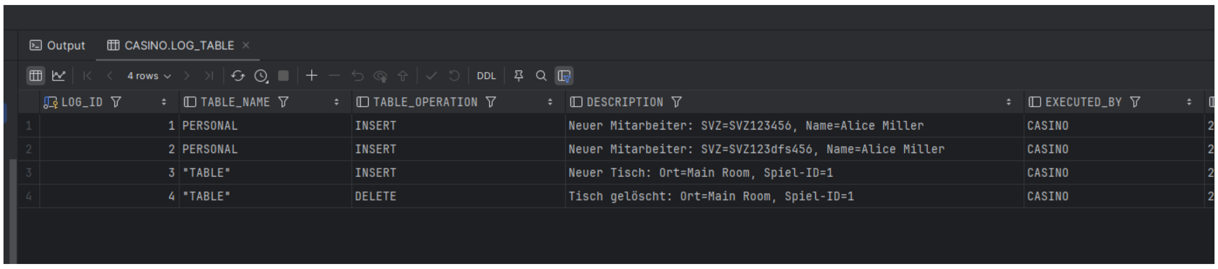
After execution:



ID	PERSONAL	"table"	"date"	START_TIME	END_TIME
1	1	1	2024-01-01	2024-01-01 08:00:00.000000	2024-01-01 16:00:00.000000
21	1	1	2025-06-02	2025-06-02 06:47:38.135500	2025-06-02 09:47:38.135500
22	1	1	2025-06-01	2025-06-01 14:00:00.000000	2025-06-01 18:00:00.000000

LOG TABLE:

Also there is a log table which captures all actions (the trigger was added during the tests so there is not everything there)



	LOG_ID	TABLE_NAME	TABLE_OPERATION	DESCRIPTION	EXECUTED_BY	
1	1	PERSONAL	INSERT	Neuer Mitarbeiter: SVZ=SVZ123456, Name=Alice Miller	CASINO	2
2	2	PERSONAL	INSERT	Neuer Mitarbeiter: SVZ=SVZ123dfs456, Name=Alice Miller	CASINO	2
3	3	"TABLE"	INSERT	Neuer Tisch: Ort=Main Room, Spiel-ID=1	CASINO	2
4	4	"TABLE"	DELETE	Tisch gelöscht: Ort=Main Room, Spiel-ID=1	CASINO	2

it loggs the actions on the tables with this schema

Script is to long to copy here. It can be found in the log_trg.sql file.

For the tests there are also testpackages available for everything besides the normal tests.