Proiect SGBD

Tavă Andrei-Daniel 233

Magazin online de produse software

1. Descriere și utilitate:

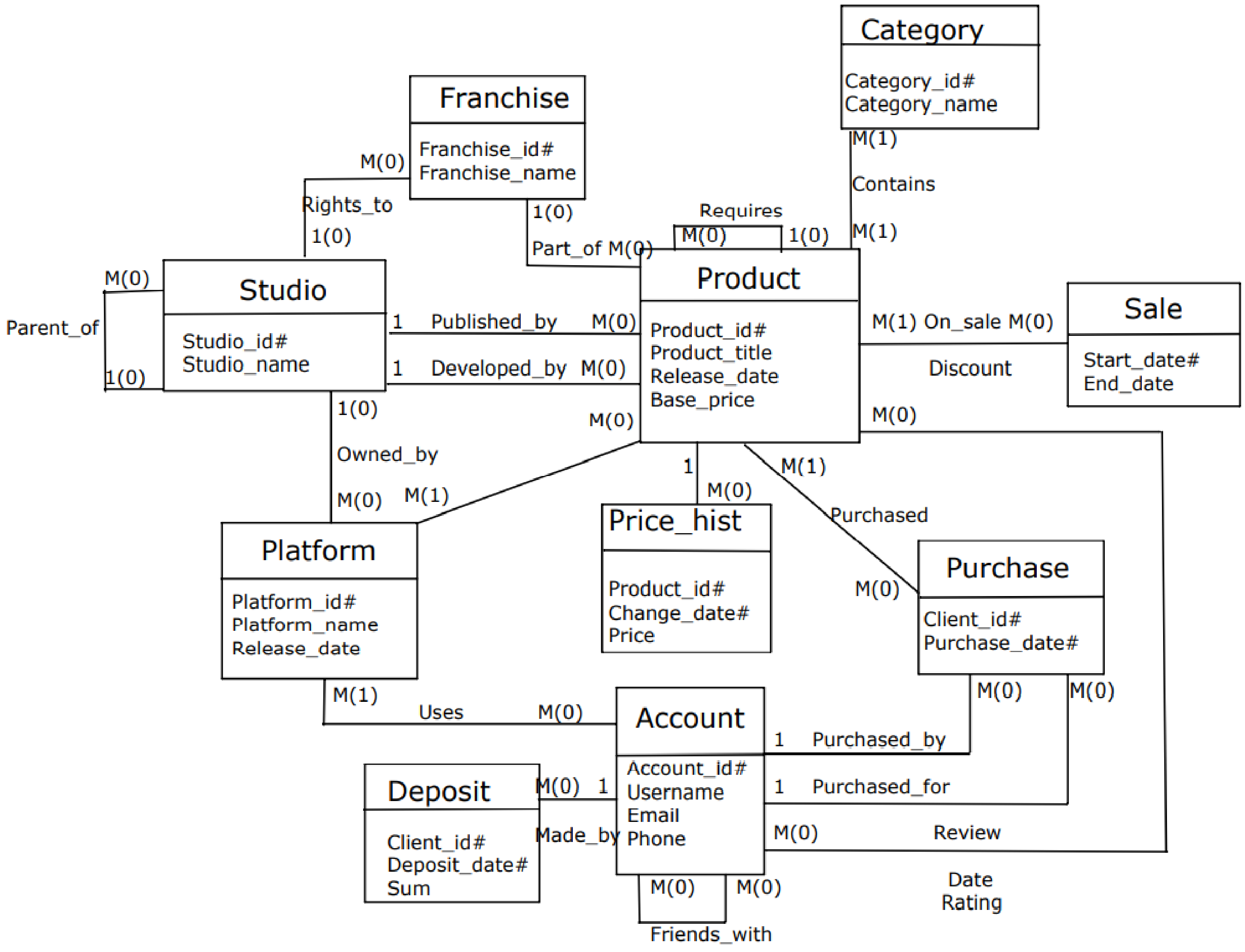
Proiectul modelează baza de date a unui magazin online de produse software(in principal jocuri video). Produsele sunt dezvoltate și publicate de studiouri pentru una sau mai multe platforme și pot face parte din diverse categorii. Utilizatorii pot cumpără unul sau mai multe produse într-o tranzacție pentru ei sau cadou pentru alt utilizator. Produsele pot fi supuse reducerilor și pot primii recenzii de la utilizatori.

Baza de date stochează informații despre produse,utilizatori,reduceri,tranzacții și recenzii. Ea poate răspunde la diverse interogări simple,complexe și într-o anumită măsură istorice.

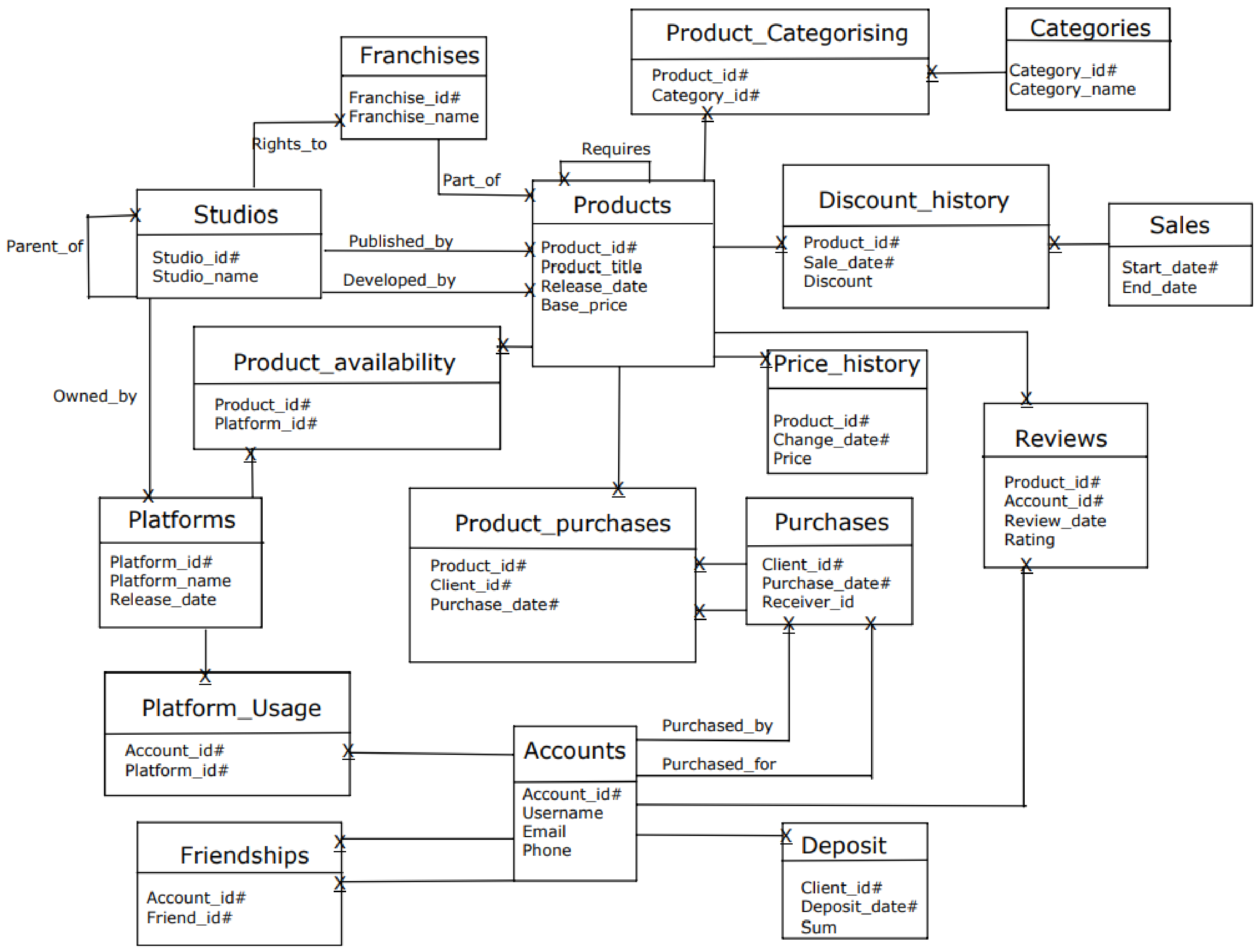
Reguli și restricții:

- un produs are un singur developer și un singur publisher.
- un produs poate face parte din mai multe categorii
- un produs este disponibil pe mai multe platforme, dar are aceleași atribute(precum preț) indiferent de platformă.
- un utilizator are acces la un produs cumpărat pe toate platformele pe care le deține.
- un utilizator are un singur email și un singur număr de telefon asociat.
- un utilizator poate cumpăra produse pentru alt utilizator(cadou).
- un produs nu poate fi returnat odată cumpărat(sau primit).
- două perioade de reduceri nu se pot suprapune în timp.

2. Diagrama E/R:



3. Diagrama Conceptuală:



4-5. Creare și Inserare:

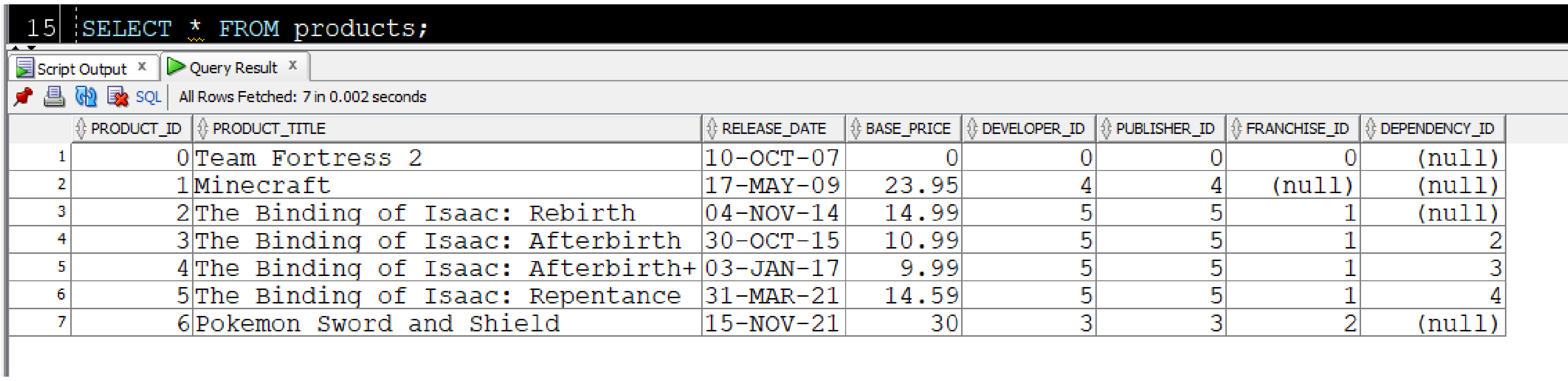
(e text, nu poză)

```
CREATE SEQUENCE account_index START WITH 0 INCREMENT BY 1 MINVALUE 0 NOCACHE;
CREATE SEQUENCE product_index START WITH 0 INCREMENT BY 1 MINVALUE 0 NOCACHE;
CREATE SEQUENCE category_index START WITH 0 INCREMENT BY 1 MINVALUE 0 NOCACHE;
CREATE SEQUENCE platform_index START WITH 0 INCREMENT BY 1 MINVALUE 0 NOCACHE;
CREATE SEQUENCE studio_index START WITH 0 INCREMENT BY 1 MINVALUE 0 NOCACHE;
CREATE SEQUENCE franchise_index START WITH 0 INCREMENT BY 1 MINVALUE 0 NOCACHE;
/*********************************************************************************/
CREATE TABLE categories (
    category_id     NUMBER(5) DEFAULT category_index.NEXTVAL PRIMARY KEY,
    category_name   VARCHAR2(32) NOT NULL UNIQUE);

CREATE TABLE studios (
    studio_id     NUMBER(5) DEFAULT studio_index.NEXTVAL PRIMARY KEY,
    studio_name   VARCHAR2(64) NOT NULL UNIQUE,
    parent_id     NUMBER(5) REFERENCES studios ( studio_id ) ON DELETE SET NULL);

CREATE TABLE franchises (
    franchise_id     NUMBER(5) DEFAULT franchise_index.NEXTVAL PRIMARY KEY,
    franchise_name   VARCHAR2(64) NOT NULL UNIQUE,
    holder_id        NUMBER(5) REFERENCES studios ( studio_id ) ON DELETE SET NULL);

CREATE TABLE platforms (
    platform_id     NUMBER(5) DEFAULT platform_index.NEXTVAL PRIMARY KEY,
    platform_name   VARCHAR2(32) NOT NULL UNIQUE,
    release_date    DATE DEFAULT sysdate,
    owner_id        NUMBER(5) REFERENCES studios ( studio_id ) ON DELETE SET NULL);

CREATE TABLE sales (
    start_date   DATE DEFAULT sysdate PRIMARY KEY,
    end_date     DATE NOT NULL,
    sale_name    VARCHAR2(64) NOT NULL,
    CONSTRAINT sales_valid CHECK ( end_date > start_date ));

CREATE TABLE accounts (
    account_id      NUMBER(5) DEFAULT account_index.NEXTVAL PRIMARY KEY,
    username        VARCHAR2(32) NOT NULL UNIQUE,
    display_name    VARCHAR2(64) NOT NULL,
    join_date       DATE DEFAULT sysdate NOT NULL,
    email           VARCHAR2(32) NOT NULL UNIQUE CHECK ( email LIKE '%@%' ),
    phone           VARCHAR2(15) UNIQUE,
    password_hash   CHAR(64) NOT NULL);

CREATE TABLE products (
    product_id      NUMBER(5) DEFAULT product_index.NEXTVAL PRIMARY KEY,
    product_title   VARCHAR2(64) NOT NULL UNIQUE,
    release_date    DATE DEFAULT sysdate,
    base_price      NUMBER(4, 2) DEFAULT 0 NOT NULL CHECK ( base_price >= 0 ),
    developer_id    NUMBER(5) NOT NULL REFERENCES studios ( studio_id ) ON DELETE CASCADE,
    publisher_id    NUMBER(5) NOT NULL REFERENCES studios ( studio_id ) ON DELETE CASCADE,
    franchise_id    NUMBER(5) REFERENCES franchises ( franchise_id ) ON DELETE SET NULL,
    dependency_id   NUMBER(5) REFERENCES products ( product_id ) ON DELETE CASCADE);

CREATE TABLE price_history (
    product_id      NUMBER(5) REFERENCES products (product_id) ON DELETE CASCADE,
    change_date     DATE DEFAULT sysdate,
    price           NUMBER(4,2) DEFAULT 0 NOT NULL CHECK ( price >= 0),
    CONSTRAINT pk_price_history PRIMARY KEY ( product_id, change_date ));

CREATE TABLE purchases (
    client_id       NUMBER(5) REFERENCES accounts ( account_id ) ON DELETE CASCADE,
    purchase_date   DATE DEFAULT sysdate,
    receiver_id     NUMBER(5) REFERENCES accounts ( account_id ) ON DELETE SET NULL,
    CONSTRAINT pk_purchases PRIMARY KEY ( client_id, purchase_date ));

CREATE TABLE deposits (
    client_id NUMBER(5) REFERENCES accounts (account_id) ON DELETE CASCADE,
    deposit_date DATE DEFAULT sysdate,
    deposit_sum NUMBER(6,2) NOT NULL,
```

```sql
    CONSTRAINT pk_deposits PRIMARY KEY ( client_id, deposit_date ));

CREATE TABLE reviews (
    account_id    NUMBER(5) REFERENCES accounts ( account_id ) ON DELETE CASCADE,
    product_id    NUMBER(5) REFERENCES products ( product_id ) ON DELETE CASCADE,
    review_date   DATE DEFAULT sysdate NOT NULL,
    rating        NUMBER(3, 2) NOT NULL CHECK ( rating >= 0 AND rating <= 5 ),
    CONSTRAINT pk_reviews PRIMARY KEY ( account_id,product_id ));

CREATE TABLE product_categorising (
    product_id    NUMBER(5) REFERENCES products ( product_id ) ON DELETE CASCADE,
    category_id   NUMBER(5) REFERENCES categories ( category_id ) ON DELETE CASCADE,
    CONSTRAINT pk_prod_cat PRIMARY KEY ( product_id,category_id ));

CREATE TABLE product_availability (
    product_id    NUMBER(5) REFERENCES products ( product_id ) ON DELETE CASCADE,
    platform_id   NUMBER(5) REFERENCES platforms ( platform_id ) ON DELETE CASCADE,
    CONSTRAINT pk_prod_plat PRIMARY KEY ( product_id,platform_id ));

CREATE TABLE discount_history (
    product_id   NUMBER(5) REFERENCES products ( product_id ) ON DELETE CASCADE,
    sale_date    DATE REFERENCES sales ( start_date ) ON DELETE CASCADE,
    discount     NUMBER(2, 2) NOT NULL CHECK ( discount > 0 AND discount <= 1 ),
    CONSTRAINT pk_discount PRIMARY KEY ( product_id,sale_date ));

CREATE TABLE platform_usage (
    account_id    NUMBER(5) REFERENCES accounts ( account_id ) ON DELETE CASCADE,
    platform_id   NUMBER(5) REFERENCES platforms ( platform_id ) ON DELETE CASCADE,
    CONSTRAINT pk_plat_usg PRIMARY KEY ( platform_id,account_id ));

CREATE TABLE friendships (
    account_id   NUMBER(5) REFERENCES accounts ( account_id ) ON DELETE CASCADE,
    friend_id    NUMBER(5) REFERENCES accounts ( account_id ) ON DELETE CASCADE,
    CONSTRAINT non_self_friend CHECK ( account_id != friend_id ),
    CONSTRAINT pk_friends PRIMARY KEY ( account_id,friend_id ));

CREATE TABLE product_purchases (
    product_id      NUMBER(5) REFERENCES products ( product_id ) ON DELETE CASCADE,
    client_id       NUMBER(5),
    purchase_date   DATE,
    CONSTRAINT fk_prod_purch FOREIGN KEY ( client_id,purchase_date ) REFERENCES purchases ( client_id,purchase_date )
ON DELETE CASCADE,
    CONSTRAINT pk_prod_purch PRIMARY KEY ( product_id,client_id,purchase_date ));
/**********************************************/

INSERT INTO accounts (username,display_name,email,phone,password_hash,join_date)
VALUES ('widderr','widz','tavaandrei@gmail.com','+40759145680',
    '7138f2e1e38c8b5b9e06d4822e083560d4ce717b8c45f571b6768d852193f0d7',
    TO_DATE('07/06/2015, 7:27:27 PM', 'MM/DD/YYYY, HH12:MI:SS AM'));

INSERT INTO accounts (username,display_name,email,phone,password_hash,join_date)
VALUES ('berkesmcheru','bigboiberke','berkemusellim@hotmail.com','+40757049004',
    '3c97be15cc5259a68287081c4b41d7ef0cfea261edc9dcbca2b2357a737c34ca',
    TO_DATE('05/07/2020, 5:26:26 PM', 'MM/DD/YYYY, HH12:MI:SS AM'));

INSERT INTO accounts (username,display_name,email,phone,password_hash,join_date)
VALUES ('Qmpz','Diaconu','weAre@palmier.com','(void*(0))',
    'af5f269ddf697cd26239e7f7e6853e1d3e8fdcd213b9f0ffe825f7725582643f',
    TO_DATE('05/07/2020, 5:30:05 PM', 'MM/DD/YYYY, HH12:MI:SS AM'));

INSERT INTO accounts (username,display_name,email,phone,password_hash,join_date)
VALUES ('JohnXina','JohnCena','youcantseeme@fbi.mail.us',NULL,
    'c83f0be82792393aa49eaae8115931279c0d45259577acea04e50d3b4b7b0344',
    TO_DATE('05/07/2020, 5:39:54 PM', 'MM/DD/YYYY, HH12:MI:SS AM'));

INSERT INTO accounts (username,display_name,email,phone,password_hash,join_date)
VALUES ('freesciofficial','widz','sizzlefrostindeed@gmail.com',NULL,
    'f0cc9b7bf0cb92e5bca6d191a7a4350f17f3ea0d28e0a5e143b347ea560a4434',
    TO_DATE('05/09/2015, 5:30:11 PM', 'MM/DD/YYYY, HH12:MI:SS AM'));

INSERT INTO accounts (username,display_name,email,phone,password_hash,join_date)
VALUES ('popescu_d017','Decebal Popescu','decebalpopescu2013@yahoo.ro','074000000',
    '05047861e93fb4b8ce12534d7b4eb21020595c45dcf2693bfe906da1b4b20fc5',
    TO_DATE('07/10/2021, 2:15:11 PM', 'MM/DD/YYYY, HH12:MI:SS AM'));
/***********************************************/
INSERT INTO categories ( category_name ) VALUES ( 'Roguelike' );
INSERT INTO categories ( category_name ) VALUES ( 'First Person Shooter' );
INSERT INTO categories ( category_name ) VALUES ( 'Multiplayer' );
INSERT INTO categories ( category_name ) VALUES ( 'Singleplayer' );
INSERT INTO categories ( category_name ) VALUES ( 'Sandbox' );
INSERT INTO categories ( category_name ) VALUES ( '2D' );
INSERT INTO categories ( category_name ) VALUES ( '3D' );
/***********************************************/
INSERT INTO studios (studio_name,parent_id)
VALUES ('Valve',NULL);

INSERT INTO studios (studio_name,parent_id)
VALUES ('Sony Interactive Entertainment',NULL);

INSERT INTO studios (studio_name,parent_id)
VALUES ('Microsoft',NULL);

INSERT INTO studios (studio_name,parent_id)
VALUES ('Nintendo',NULL);

INSERT INTO studios (studio_name,parent_id)
VALUES ('Mojang',2);

INSERT INTO studios (studio_name,parent_id)
VALUES ('Nicalis, Inc',NULL);
/***********************************************/
INSERT INTO platforms (platform_name,owner_id,release_date)
VALUES ('Personal Computer',NULL,NULL);

INSERT INTO platforms (platform_name,owner_id,release_date)
VALUES ('PlayStation 3',1,TO_DATE('23 MAR 2007', 'DD MON YYYY'));

INSERT INTO platforms (platform_name,owner_id,release_date)
VALUES ('PlayStation 4',1,TO_DATE('29 NOV 2013', 'DD MON YYYY'));

INSERT INTO platforms (platform_name,owner_id,release_date)
VALUES ('PlayStation 5',1,TO_DATE('19 NOV 2020', 'DD MON YYYY'));

INSERT INTO platforms (platform_name,owner_id,release_date)
VALUES ('Xbox 360',2,TO_DATE('02 DEC 2005', 'DD MON YYYY'));

INSERT INTO platforms (platform_name,owner_id,release_date)
VALUES ('Xbox One',2,TO_DATE('22 NOV 2013', 'DD MON YYYY'));

INSERT INTO platforms (platform_name,owner_id,release_date)
VALUES ('Xbox Series X',2,TO_DATE('10 NOV 2020', 'DD MON YYYY'));

INSERT INTO platforms (platform_name,owner_id,release_date)
VALUES ('Nintendo Switch',3,TO_DATE('03 MAR 2017', 'DD MON YYYY'));

INSERT INTO platforms (platform_name,owner_id,release_date)
VALUES ('Nintendo Gamecube',3,TO_DATE('14 SEP 2001', 'DD MON YYYY'));
/***********************************************/

INSERT INTO franchises (franchise_name,holder_id)
VALUES ('Team Fortress',0);

INSERT INTO franchises (franchise_name,holder_id)
VALUES ('Binding of Isaac',5);

INSERT INTO franchises (franchise_name,holder_id)
VALUES ('Pokemon',3);

INSERT INTO franchises (franchise_name,holder_id)
VALUES ('Alice in Wonderland',NULL);

INSERT INTO franchises (franchise_name,holder_id)
VALUES ('Age of Empires',2);
```

```
/***********************************************/
INSERT INTO products (product_title,release_date,base_price,publisher_id,developer_id,franchise_id,dependency_id)
VALUES ('Team Fortress 2',TO_DATE('10 OCT 2007', 'DD MON YYYY'),0,0,0,0,NULL);

INSERT INTO products (product_title,release_date,base_price,publisher_id,developer_id,franchise_id,dependency_id)
VALUES ('Minecraft',TO_DATE('17 MAY 2009', 'DD MON YYYY'),23.95,4,4,NULL,NULL);

INSERT INTO products (product_title,release_date,base_price,publisher_id,developer_id,franchise_id,dependency_id)
VALUES ('The Binding of Isaac: Rebirth',TO_DATE('04 NOV 2014', 'DD MON YYYY'),14.99,5,5,1,NULL);

INSERT INTO products (product_title,release_date,base_price,publisher_id,developer_id,franchise_id,dependency_id)
VALUES ('The Binding of Isaac: Afterbirth',TO_DATE('30 OCT 2015', 'DD MON YYYY'),10.99,5,5,1,2);

INSERT INTO products (product_title,release_date,base_price,publisher_id,developer_id,franchise_id,dependency_id)
VALUES ('The Binding of Isaac: Afterbirth+',TO_DATE('03 JAN 2017', 'DD MON YYYY'),9.99,5,5,1,3);

INSERT INTO products (product_title,release_date,base_price,publisher_id,developer_id,franchise_id,dependency_id)
VALUES ('The Binding of Isaac: Repentance',TO_DATE('31 MAR 2021', 'DD MON YYYY'),14.59,5,5,1,4);

INSERT INTO products (product_title,release_date,base_price,publisher_id,developer_id,franchise_id,dependency_id)
VALUES ('Pokemon Sword and Shield',TO_DATE('15 NOV 2021', 'DD MON YYYY'),30,3,3,2,NULL);

/***********************************************/
INSERT INTO price_history (product_id, change_date, price)
VALUES (1,TO_DATE('07 JAN 2018', 'DD MON YYYY'), 16.99);

INSERT INTO price_history (product_id, change_date, price)
VALUES (1,TO_DATE('23 MAR 2020', 'DD MON YYYY'), 20);

INSERT INTO price_history (product_id, change_date, price)
VALUES (0,TO_DATE('03 APR 2015', 'DD MON YYYY'), 20);

INSERT INTO price_history (product_id, change_date, price)
VALUES (2,TO_DATE('29 JUL 2017', 'DD MON YYYY'), 19.99);

INSERT INTO price_history (product_id, change_date, price)
VALUES (3,TO_DATE('31 MAR 2021', 'DD MON YYYY'), 15.99);

/***********************************************/
INSERT INTO product_categorising VALUES (0,1);
INSERT INTO product_categorising VALUES (0,2);
INSERT INTO product_categorising VALUES (0,6);
INSERT INTO product_categorising VALUES (1,2);
INSERT INTO product_categorising VALUES (1,3);
INSERT INTO product_categorising VALUES (1,4);
INSERT INTO product_categorising VALUES (1,6);
INSERT INTO product_categorising VALUES (2,0);
INSERT INTO product_categorising VALUES (2,3);
INSERT INTO product_categorising VALUES (2,5);
INSERT INTO product_categorising VALUES (3,0);
INSERT INTO product_categorising VALUES (3,3);
INSERT INTO product_categorising VALUES (3,5);
INSERT INTO product_categorising VALUES (4,0);
INSERT INTO product_categorising VALUES (4,3);
INSERT INTO product_categorising VALUES (4,5);
INSERT INTO product_categorising VALUES (5,0);
INSERT INTO product_categorising VALUES (5,3);
INSERT INTO product_categorising VALUES (5,5);
INSERT INTO product_categorising VALUES (6,3);
INSERT INTO product_categorising VALUES (6,6);
/***********************************************/
INSERT INTO product_availability VALUES (0,0);
INSERT INTO product_availability VALUES (0,1);
INSERT INTO product_availability VALUES (0,4);
INSERT INTO product_availability VALUES (1,0);
INSERT INTO product_availability VALUES (1,1);
INSERT INTO product_availability VALUES (1,2);
INSERT INTO product_availability VALUES (1,3);
INSERT INTO product_availability VALUES (1,4);
INSERT INTO product_availability VALUES (1,5);
INSERT INTO product_availability VALUES (1,6);
INSERT INTO product_availability VALUES (1,7);
INSERT INTO product_availability VALUES (2,0);
INSERT INTO product_availability VALUES (2,2);
INSERT INTO product_availability VALUES (2,5);
INSERT INTO product_availability VALUES (3,0);
INSERT INTO product_availability VALUES (3,2);
INSERT INTO product_availability VALUES (3,5);
INSERT INTO product_availability VALUES (4,0);
INSERT INTO product_availability VALUES (4,2);
INSERT INTO product_availability VALUES (4,5);
INSERT INTO product_availability VALUES (5,0);
INSERT INTO product_availability VALUES (5,2);
INSERT INTO product_availability VALUES (5,5);
INSERT INTO product_availability VALUES (6,7);
/***************************************************************************************************/
INSERT INTO sales VALUES (TO_DATE('01 AUG 2018', 'DD MON YYYY'),TO_DATE('02 AUG 2018', 'DD MON YYYY'),'Lightning
Sale');
INSERT INTO sales VALUES (TO_DATE('28 DEC 2019', 'DD MON YYYY'),TO_DATE('07 JAN 2020', 'DD MON YYYY'),'Winter Sale
2020');
INSERT INTO sales VALUES (TO_DATE('31 MAR 2021', 'DD MON YYYY'),TO_DATE('07 APR 2021', 'DD MON YYYY'),'Roguelike Sale
2021');
INSERT INTO sales VALUES (TO_DATE('15 NOV 2021', 'DD MON YYYY'),TO_DATE('20 NOV 2021', 'DD MON YYYY'),'Nintendo
Handheld Sale');
INSERT INTO sales VALUES (TO_DATE('19 JUN 2022', 'DD MON YYYY'),TO_DATE('30 JUN 2022', 'DD MON YYYY'),'Summer Sale
2022');
INSERT INTO sales VALUES (TO_DATE('01 JAN 2023', 'DD MON YYYY'),TO_DATE('01 JAN 2024', 'DD MON YYYY'),'The Sale To End
All Sales');
/***********************************************/
INSERT INTO discount_history VALUES (1,TO_DATE('01 AUG 2018', 'DD MON YYYY'),0.3);
INSERT INTO discount_history VALUES (1,TO_DATE('28 DEC 2019', 'DD MON YYYY'),0.15);
INSERT INTO discount_history VALUES (2,TO_DATE('28 DEC 2019', 'DD MON YYYY'),0.2);
INSERT INTO discount_history VALUES (3,TO_DATE('28 DEC 2019', 'DD MON YYYY'),0.25);
INSERT INTO discount_history VALUES (4,TO_DATE('28 DEC 2019', 'DD MON YYYY'),0.1);
INSERT INTO discount_history VALUES (2,TO_DATE('31 MAR 2021', 'DD MON YYYY'),0.5);
INSERT INTO discount_history VALUES (3,TO_DATE('31 MAR 2021', 'DD MON YYYY'),0.5);
INSERT INTO discount_history VALUES (4,TO_DATE('31 MAR 2021', 'DD MON YYYY'),0.5);
INSERT INTO discount_history VALUES (5,TO_DATE('31 MAR 2021', 'DD MON YYYY'),0.5);
INSERT INTO discount_history VALUES (6,TO_DATE('15 NOV 2021', 'DD MON YYYY'),0.5);
INSERT INTO discount_history VALUES (1,TO_DATE('19 JUN 2022', 'DD MON YYYY'),0.1);
INSERT INTO discount_history VALUES (2,TO_DATE('19 JUN 2022', 'DD MON YYYY'),0.2);
/*******************************************************************************************************/
INSERT INTO reviews VALUES (0,0,TO_DATE('08/12/2015', 'DD/MM/YYYY'),4.7);
INSERT INTO reviews VALUES (0,1,TO_DATE('07/07/2018', 'DD/MM/YYYY'),5);
INSERT INTO reviews VALUES (0,5,TO_DATE('15/09/2021', 'DD/MM/YYYY'),4);
INSERT INTO reviews VALUES (1,1,TO_DATE('08/07/2018', 'DD/MM/YYYY'),5);
INSERT INTO reviews VALUES (1,6,TO_DATE('17/11/2021', 'DD/MM/YYYY'),3);
INSERT INTO reviews VALUES (3,1,TO_DATE('10/07/2018', 'DD/MM/YYYY'),5);
INSERT INTO reviews VALUES (4,1,TO_DATE('11/07/2018', 'DD/MM/YYYY'),5);
INSERT INTO reviews VALUES (5,1,TO_DATE('12/07/2018', 'DD/MM/YYYY'),5);
INSERT INTO reviews VALUES (5,2,TO_DATE('21/03/2022', 'DD/MM/YYYY'),1.13);
/***********************************************/
INSERT INTO purchases (client_id,receiver_id,purchase_date)
VALUES (0,0,TO_DATE('08/07/2015', 'DD/MM/YYYY'));

INSERT INTO purchases (client_id,receiver_id,purchase_date)
VALUES (0,0,TO_DATE('02/04/2021', 'DD/MM/YYYY'));

INSERT INTO purchases (client_id,receiver_id,purchase_date)
VALUES (1,1,TO_DATE('01/08/2018', 'DD/MM/YYYY'));

INSERT INTO purchases (client_id,receiver_id,purchase_date)
VALUES (1,1,TO_DATE('16/11/2021', 'DD/MM/YYYY'));

INSERT INTO purchases (client_id,receiver_id,purchase_date)
VALUES (3,3,TO_DATE('03/08/2018', 'DD/MM/YYYY'));

INSERT INTO purchases (client_id,receiver_id,purchase_date)
VALUES (4,4,TO_DATE('07/07/2015', 'DD/MM/YYYY'));

INSERT INTO purchases (client_id,receiver_id,purchase_date)
VALUES (5,0,TO_DATE('19/11/2017', 'DD/MM/YYYY'));
/***********************************************/
INSERT INTO deposits (client_id,deposit_date,deposit_sum)
```

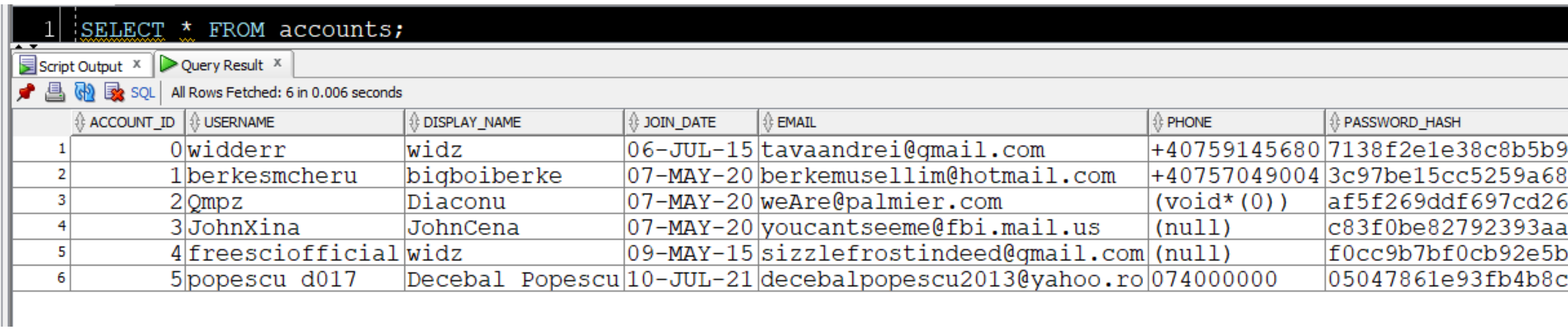
```
VALUES (0,TO_DATE('08/07/2015', 'DD/MM/YYYY'),100);

INSERT INTO deposits (client_id,deposit_date,deposit_sum)
VALUES (0,TO_DATE('01/04/2021', 'DD/MM/YYYY'),100);

INSERT INTO deposits (client_id,deposit_date,deposit_sum)
VALUES (1,TO_DATE('01/08/2018', 'DD/MM/YYYY'),50);

INSERT INTO deposits (client_id,deposit_date,deposit_sum)
VALUES (3,TO_DATE('01/08/2018', 'DD/MM/YYYY'),25);

INSERT INTO deposits (client_id,deposit_date,deposit_sum)
VALUES (4,TO_DATE('07/07/2015', 'DD/MM/YYYY'),1);

INSERT INTO deposits (client_id,deposit_date,deposit_sum)
VALUES (5,TO_DATE('19/11/2017', 'DD/MM/YYYY'),35);
/***********************************************/

INSERT INTO product_purchases (client_id,purchase_date,product_id)
VALUES (0,TO_DATE('08/07/2015', 'DD/MM/YYYY'),0);

INSERT INTO product_purchases (client_id,purchase_date,product_id)
VALUES (0,TO_DATE('02/04/2021', 'DD/MM/YYYY'),2);

INSERT INTO product_purchases (client_id,purchase_date,product_id)
VALUES (0,TO_DATE('02/04/2021', 'DD/MM/YYYY'),3);

INSERT INTO product_purchases (client_id,purchase_date,product_id)
VALUES (0,TO_DATE('02/04/2021', 'DD/MM/YYYY'),4);

INSERT INTO product_purchases (client_id,purchase_date,product_id)
VALUES (0,TO_DATE('02/04/2021', 'DD/MM/YYYY'),5);

INSERT INTO product_purchases (client_id,purchase_date,product_id)
VALUES (1,TO_DATE('01/08/2018', 'DD/MM/YYYY'),1);

INSERT INTO product_purchases (client_id,purchase_date,product_id)
VALUES (1,TO_DATE('16/11/2021', 'DD/MM/YYYY'),6);

INSERT INTO product_purchases (client_id,purchase_date,product_id)
VALUES (3,TO_DATE('03/08/2018', 'DD/MM/YYYY'),1);

INSERT INTO product_purchases (client_id,purchase_date,product_id)
VALUES (4,TO_DATE('07/07/2015', 'DD/MM/YYYY'),0);

INSERT INTO product_purchases (client_id,purchase_date,product_id)
VALUES (5,TO_DATE('19/11/2017', 'DD/MM/YYYY'),1);

/***********************************************/
INSERT INTO platform_usage VALUES (0,0);
INSERT INTO platform_usage VALUES (0,2);
INSERT INTO platform_usage VALUES (0,3);
INSERT INTO platform_usage VALUES (1,0);
INSERT INTO platform_usage VALUES (1,7);
INSERT INTO platform_usage VALUES (2,0);
INSERT INTO platform_usage VALUES (3,0);
INSERT INTO platform_usage VALUES (4,0);
INSERT INTO platform_usage VALUES (5,0);
INSERT INTO platform_usage VALUES (5,2);
INSERT INTO platform_usage VALUES (5,3);
/***********************************************/
INSERT INTO friendships VALUES (0,1);
INSERT INTO friendships VALUES (0,2);
INSERT INTO friendships VALUES (0,3);
INSERT INTO friendships VALUES (0,4);
INSERT INTO friendships VALUES (0,5);
INSERT INTO friendships VALUES (1,2);
INSERT INTO friendships VALUES (1,3);
INSERT INTO friendships VALUES (2,5);
/****************************************************/
COMMIT;
```



```sql
SELECT * FROM products;
```

All Rows Fetched: 7 in 0.002 seconds

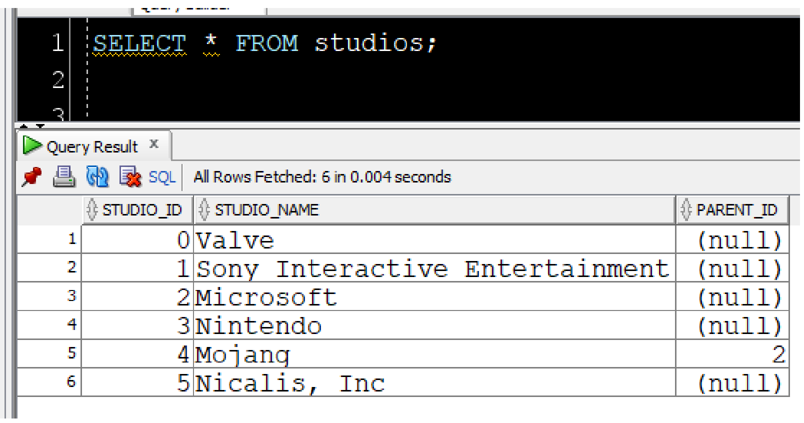
	PRODUCT_ID	PRODUCT_TITLE	RELEASE_DATE	BASE_PRICE	DEVELOPER_ID	PUBLISHER_ID	FRANCHISE_ID	DEPENDENCY_ID
1	0	Team Fortress 2	10-OCT-07	0	0	0	0	(null)
2	1	Minecraft	17-MAY-09	23.95	4	4	(null)	(null)
3	2	The Binding of Isaac: Rebirth	04-NOV-14	14.99	5	5	1	(null)
4	3	The Binding of Isaac: Afterbirth	30-OCT-15	10.99	5	5	1	2
5	4	The Binding of Isaac: Afterbirth+	03-JAN-17	9.99	5	5	1	3
6	5	The Binding of Isaac: Repentance	31-MAR-21	14.59	5	5	1	4
7	6	Pokemon Sword and Shield	15-NOV-21	30	3	3	2	(null)



```sql
SELECT * FROM accounts;
```

All Rows Fetched: 6 in 0.006 seconds

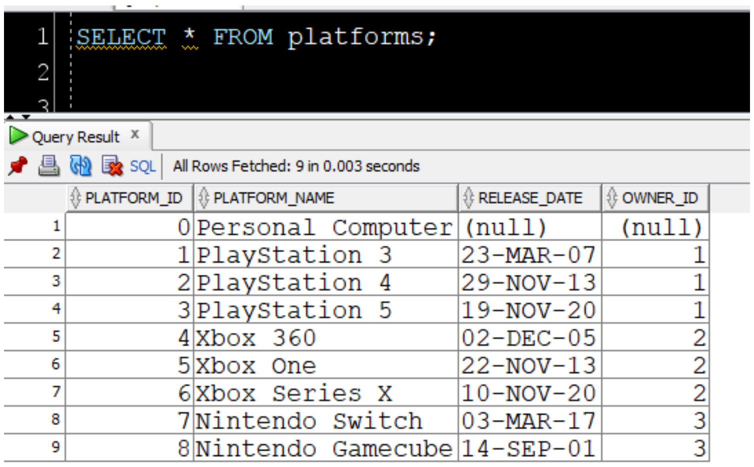
	ACCOUNT_ID	USERNAME	DISPLAY_NAME	JOIN_DATE	EMAIL	PHONE	PASSWORD_HASH
1	0	widderr	widz	06-JUL-15	tavaandrei@gmail.com	+40759145680	7138f2e1e38c8b5b9
2	1	berkesmcheru	bigboiberke	07-MAY-20	berkemusellim@hotmail.com	+40757049004	3c97be15cc5259a68
3	2	Qmpz	Diaconu	07-MAY-20	weAre@palmier.com	(void*(0))	af5f269ddf697cd26
4	3	JohnXina	JohnCena	07-MAY-20	youcantseeme@fbi.mail.us	(null)	c83f0be82792393aa
5	4	freesciofficial	widz	09-MAY-15	sizzlefrostindeed@gmail.com	(null)	f0cc9b7bf0cb92e5b
6	5	popescu d017	Decebal Popescu	10-JUL-21	decebalpopescu2013@yahoo.ro	074000000	05047861e93fb4b8c



```sql
SELECT * FROM studios;
```

All Rows Fetched: 6 in 0.004 seconds

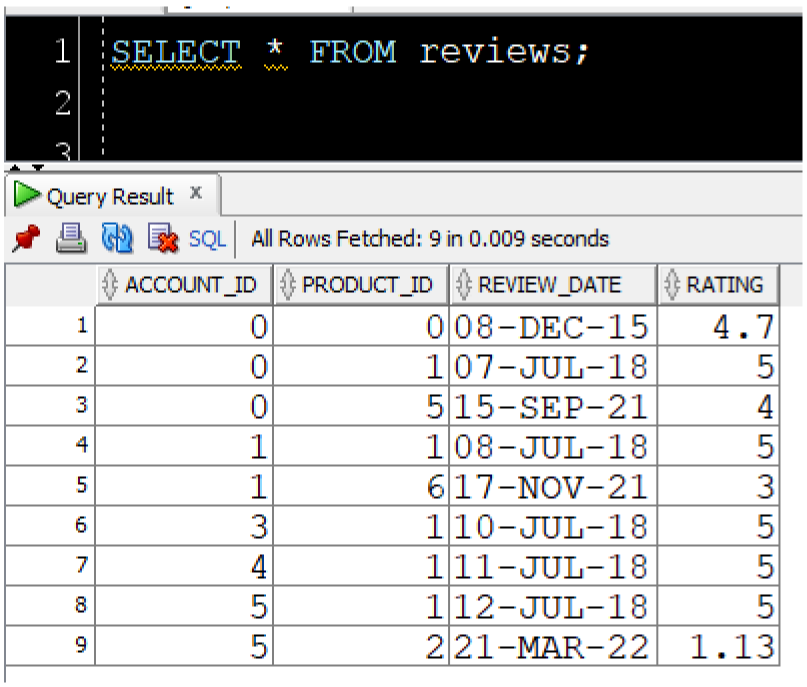
	STUDIO_ID	STUDIO_NAME	PARENT_ID
1	0	Valve	(null)
2	1	Sony Interactive Entertainment	(null)
3	2	Microsoft	(null)
4	3	Nintendo	(null)
5	4	Mojang	2
6	5	Nicalis, Inc	(null)



```sql
SELECT * FROM platforms;
```

All Rows Fetched: 9 in 0.003 seconds

	PLATFORM_ID	PLATFORM_NAME	RELEASE_DATE	OWNER_ID
1	0	Personal Computer	(null)	(null)
2	1	PlayStation 3	23-MAR-07	1
3	2	PlayStation 4	29-NOV-13	1
4	3	PlayStation 5	19-NOV-20	1
5	4	Xbox 360	02-DEC-05	2
6	5	Xbox One	22-NOV-13	2
7	6	Xbox Series X	10-NOV-20	2
8	7	Nintendo Switch	03-MAR-17	3
9	8	Nintendo Gamecube	14-SEP-01	3

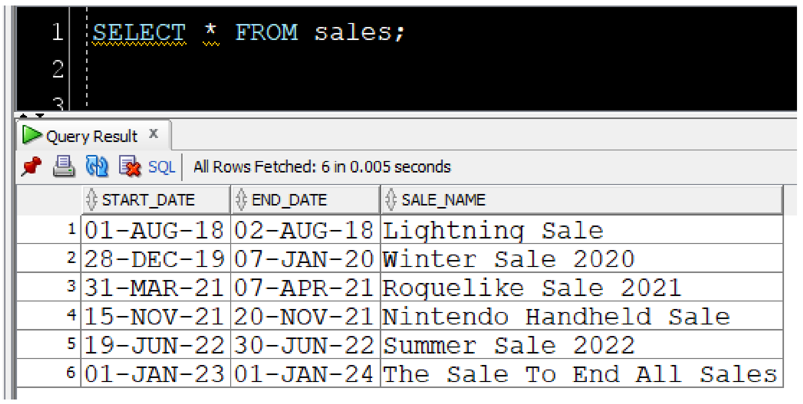


```sql
SELECT * FROM reviews;
```

Query Result

SQL | All Rows Fetched: 9 in 0.009 seconds

	ACCOUNT_ID	PRODUCT_ID	REVIEW_DATE	RATING
1	0	0	08-DEC-15	4.7
2	0	1	07-JUL-18	5
3	0	5	15-SEP-21	4
4	1	1	08-JUL-18	5
5	1	6	17-NOV-21	3
6	3	1	10-JUL-18	5
7	4	1	11-JUL-18	5
8	5	1	12-JUL-18	5
9	5	2	21-MAR-22	1.13

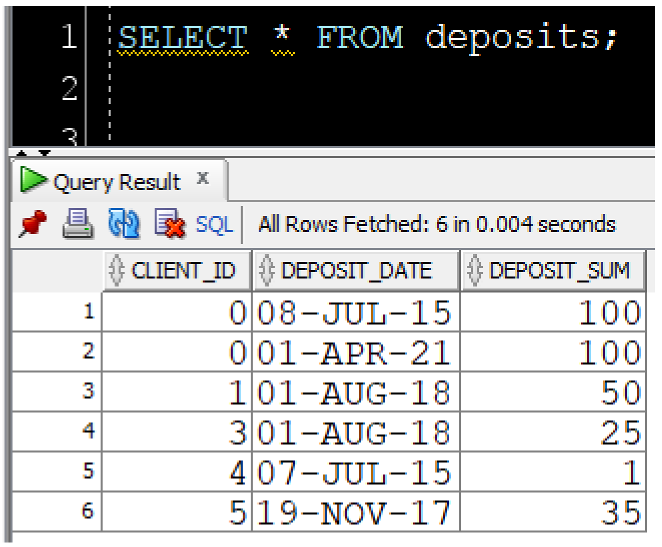


```sql
SELECT * FROM sales;
```

Query Result

SQL | All Rows Fetched: 6 in 0.005 seconds

	START_DATE	END_DATE	SALE_NAME
1	01-AUG-18	02-AUG-18	Lightning Sale
2	28-DEC-19	07-JAN-20	Winter Sale 2020
3	31-MAR-21	07-APR-21	Roguelike Sale 2021
4	15-NOV-21	20-NOV-21	Nintendo Handheld Sale
5	19-JUN-22	30-JUN-22	Summer Sale 2022
6	01-JAN-23	01-JAN-24	The Sale To End All Sales

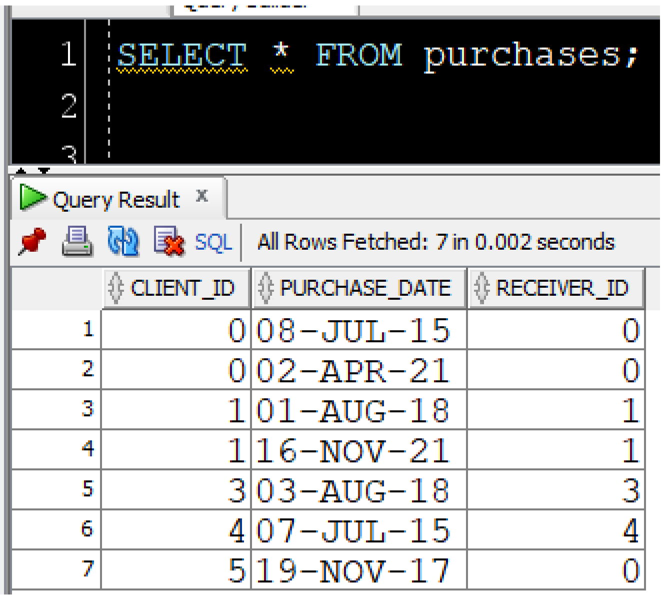


```sql
SELECT * FROM deposits;
```

Query Result

SQL | All Rows Fetched: 6 in 0.004 seconds

	CLIENT_ID	DEPOSIT_DATE	DEPOSIT_SUM
1	0	08-JUL-15	100
2	0	01-APR-21	100
3	1	01-AUG-18	50
4	3	01-AUG-18	25
5	4	07-JUL-15	1
6	5	19-NOV-17	35



```sql
SELECT * FROM purchases;
```

Query Result

SQL | All Rows Fetched: 7 in 0.002 seconds

	CLIENT_ID	PURCHASE_DATE	RECEIVER_ID
1	0	08-JUL-15	0
2	0	02-APR-21	0
3	1	01-AUG-18	1
4	1	16-NOV-21	1
5	3	03-AUG-18	3
6	4	07-JUL-15	4
7	5	19-NOV-17	0

6-9. Proceduri și Funcții:

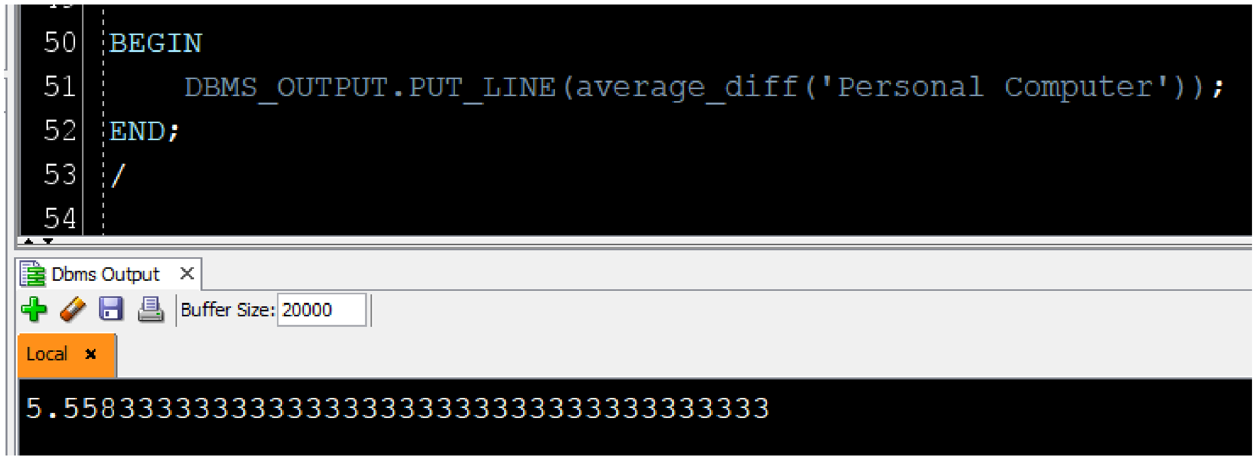
6. Funcție care să utilizele două colecții diferite

Enunț: Pentru o platforma dată, să se returneze diferența medie dintre prețul fiecărui produs și prețul celui mai scump produs din franciza respectivă.(dacă nu face parte din vreo franciza, se considera pretul maxim 0.)

Am folosit: un Index-By Table și un Nested Table

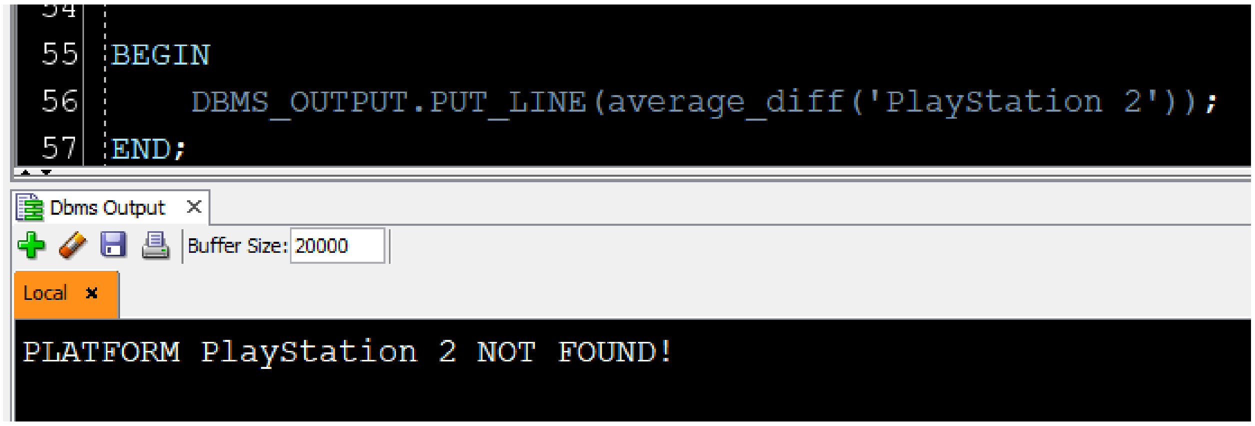
Explicație: Pentru numele platformei extragem platform_id; aici poate apărea NO_DATA_FOUND. Extragem apoi toate produsele(id,franciză,preț) într-un Nested Table. Pentru fiecare produs, verificăm dacă avem deja maximul francizei în Index-By Table și adăugăm diferența calculată. Dacă nu avem deja maximul determinat, aflăm maximul și îl stocăm în tabel. La final împărțim suma la numărul de produse, putând apărea VALUE_ERROR atunci când numărul de produse este 0.

```
FUNCTION average_diff(p_platform_name IN platforms.platform_name%TYPE) RETURN NUMBER IS
        TYPE fran_prod_map IS TABLE OF products.base_price%TYPE INDEX BY PLS_INTEGER;
        TYPE prod_rec IS RECORD
            (prod_id products.product_id%TYPE,
            fran_id products.franchise_id%TYPE,
            price products.base_price%TYPE);
        TYPE prod_tab IS TABLE OF prod_rec;
        v_products prod_tab := prod_tab();
        v_max_prod fran_prod_map;
        v_average NUMBER := 0;
        v_plat_id platforms.platform_id%TYPE;
    BEGIN
        SELECT platform_id
        INTO v_plat_id
        FROM platforms
        WHERE platform_name = p_platform_name;

        SELECT product_id,franchise_id,base_price
        BULK COLLECT INTO v_products
        FROM product_availability JOIN products USING (product_id)
        WHERE platform_id = v_plat_id;

        FOR i IN v_products.FIRST..v_products.LAST LOOP --we know for sure it's dense
           IF v_products(i).fran_id IS NULL
                THEN  v_average := v_average + v_products(i).price;
                    CONTINUE;
            END IF;

            IF NOT v_max_prod.EXISTS(v_products(i).fran_id) --if we've already computed the maximum for
this franchise
                THEN SELECT MAX(base_price)
                     INTO v_max_prod(v_products(i).fran_id)
                     FROM products
                     WHERE franchise_id = v_products(i).fran_id;
            END IF;
            v_average := v_average + v_max_prod(v_products(i).fran_id) - v_products(i).price;
        END LOOP;
        RETURN (v_average/v_products.COUNT);
    EXCEPTION
        WHEN NO_DATA_FOUND THEN DBMS_OUTPUT.PUT_LINE('PLATFORM ' || p_platform_name ||' NOT FOUND!');
                                RAISE NO_DATA_FOUND;
        WHEN VALUE_ERROR THEN DBMS_OUTPUT.PUT_LINE('THERE ARE NO PRODUCTS FOR PLATFORM ' ||
p_platform_name);
    END average_diff;
```



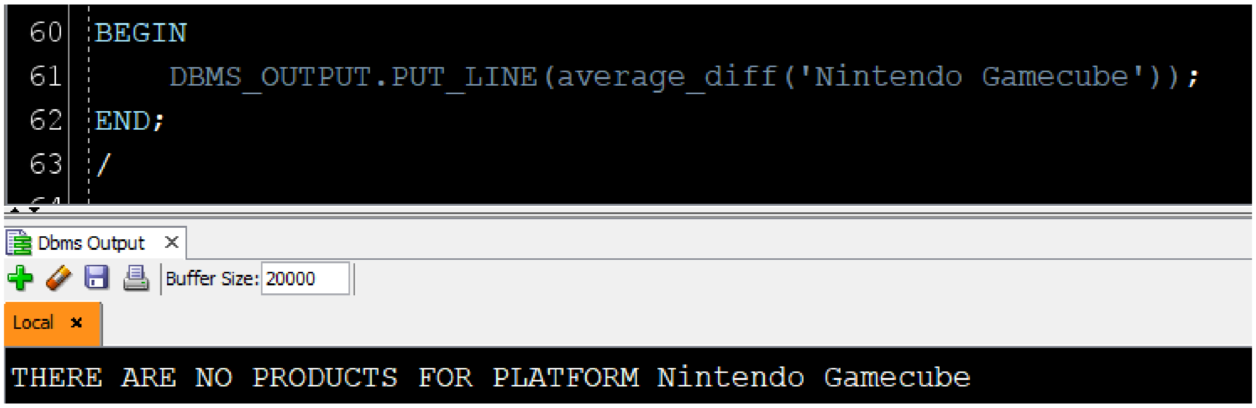
```
BEGIN
    DBMS_OUTPUT.PUT_LINE(average_diff('Personal Computer'));
END;
/
```

```
5.558333333333333333333333333333333333333
```



```
BEGIN
    DBMS_OUTPUT.PUT_LINE(average_diff('PlayStation 2'));
END;
```

```
PLATFORM PlayStation 2 NOT FOUND!
```



```
BEGIN
    DBMS_OUTPUT.PUT_LINE(average_diff('Nintendo Gamecube'));
END;
/
```

```
THERE ARE NO PRODUCTS FOR PLATFORM Nintendo Gamecube
```

7. Procedură care să folosească două tipuri de cursoare diferite

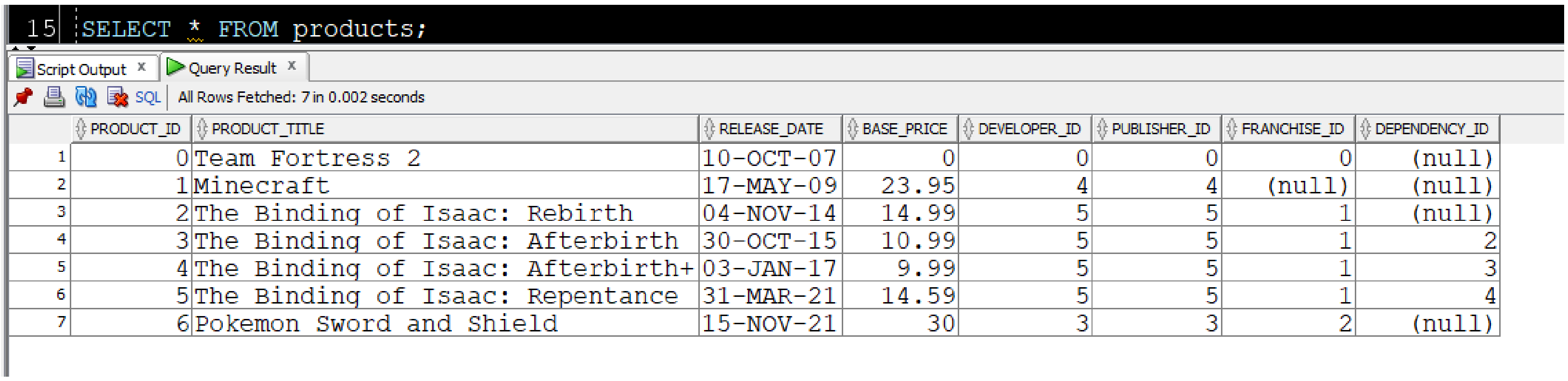
Enunț: Să se reducă cu un procent dat prețul tuturor produselor dezvoltate de către studiouri independente (care nu au alt studio parinte).

Am folosit: Cursor Parametrizat și Cursor-Subcerere

Explicație: Folosind un Cursor-Subcerere, iterez prin toate studiourile independente. Pentru fiecare studio, folosesc un Cursor Parametrizat cu parametru studio_id, pentru a itera prin toate produsele create de acel studio. Apoi doar aplic discountul pe valoarea curentă a cursorului. O Excepție este ridicată dacă valoarea cursorului nu este subunitara.

```
PROCEDURE discount_all_independent(p_discount IN products.base_price%TYPE) IS
        CURSOR products_by(studio_id studios.studio_id%TYPE) IS --parameter cursor
            SELECT product_id
            FROM products
            WHERE developer_id = studio_id
            FOR UPDATE OF base_price;
        BAD_DISCOUNT EXCEPTION;
    BEGIN
        IF p_discount NOT BETWEEN 0 AND 1 THEN RAISE BAD_DISCOUNT;
        END IF;
        FOR studio IN (SELECT studio_id     -- subquery cursor
                       FROM studios
                       WHERE parent_id IS NULL) LOOP
            FOR product IN products_by(studio.studio_id) LOOP
                UPDATE products
                SET base_price = base_price * (1 - p_discount)
                WHERE CURRENT OF products_by;
            END LOOP;
        END LOOP;
    EXCEPTION
        WHEN BAD_DISCOUNT THEN DBMS_OUTPUT.PUT_LINE('INVALID DISCOUNT VALUE');
    END discount_all_independent;
```

Înainte:

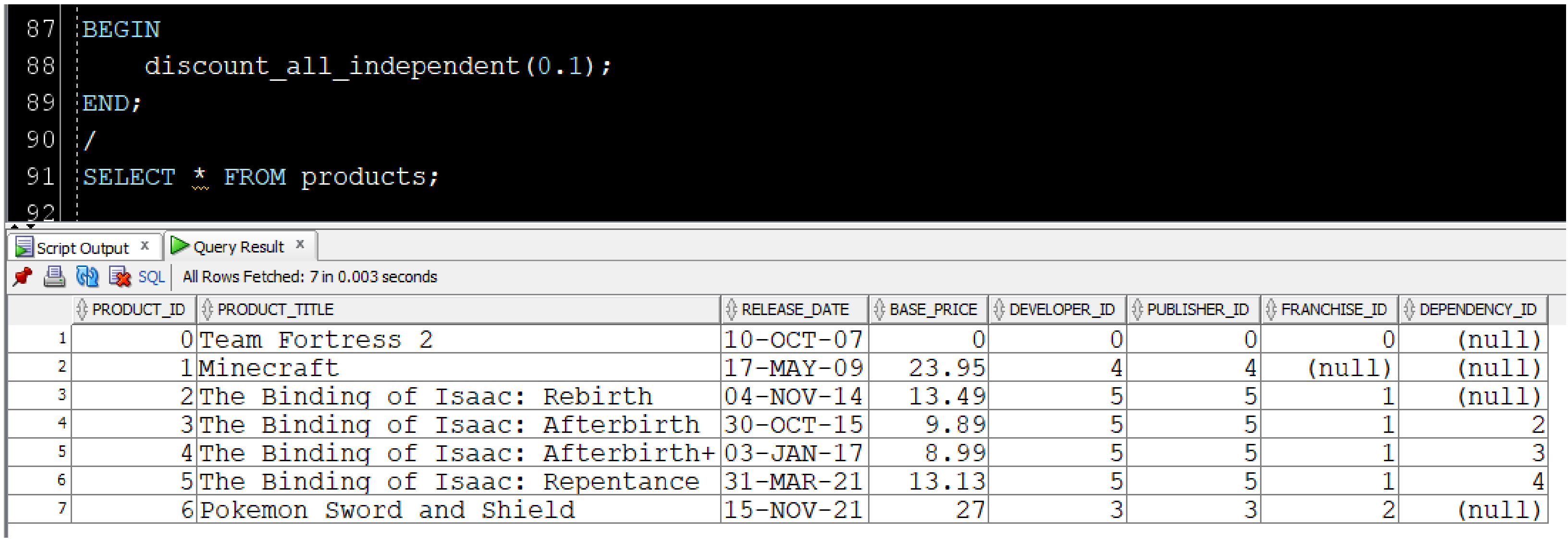


```
SELECT * FROM products;
```

All Rows Fetched: 7 in 0.002 seconds

	PRODUCT_ID	PRODUCT_TITLE	RELEASE_DATE	BASE_PRICE	DEVELOPER_ID	PUBLISHER_ID	FRANCHISE_ID	DEPENDENCY_ID
1	0	Team Fortress 2	10-OCT-07	0	0	0	0	(null)
2	1	Minecraft	17-MAY-09	23.95	4	4	(null)	(null)
3	2	The Binding of Isaac: Rebirth	04-NOV-14	14.99	5	5	1	(null)
4	3	The Binding of Isaac: Afterbirth	30-OCT-15	10.99	5	5	1	2
5	4	The Binding of Isaac: Afterbirth+	03-JAN-17	9.99	5	5	1	3
6	5	The Binding of Isaac: Repentance	31-MAR-21	14.59	5	5	1	4
7	6	Pokemon Sword and Shield	15-NOV-21	30	3	3	2	(null)

După:



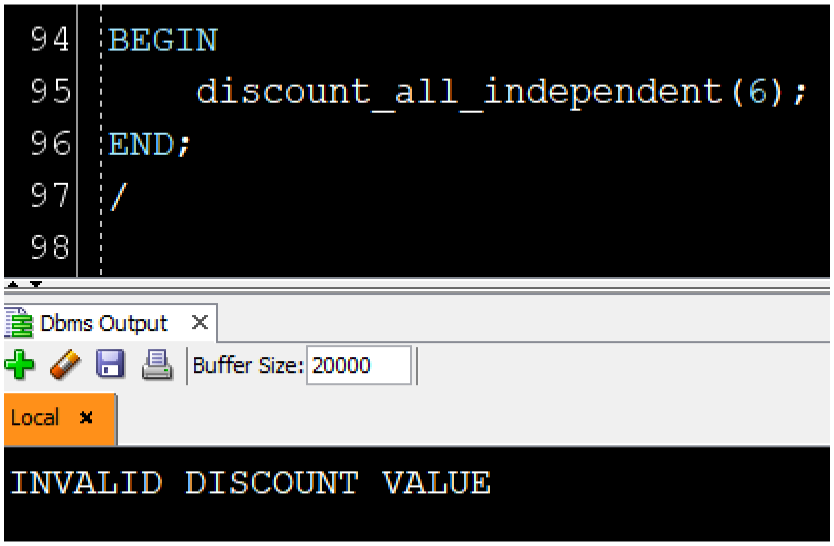
```
BEGIN
    discount_all_independent(0.1);
END;
/
SELECT * FROM products;
```

Script Output × Query Result ×

All Rows Fetched: 7 in 0.003 seconds

	PRODUCT_ID	PRODUCT_TITLE	RELEASE_DATE	BASE_PRICE	DEVELOPER_ID	PUBLISHER_ID	FRANCHISE_ID	DEPENDENCY_ID
1	0	Team Fortress 2	10-OCT-07	0	0	0	0	(null)
2	1	Minecraft	17-MAY-09	23.95	4	4	(null)	(null)
3	2	The Binding of Isaac: Rebirth	04-NOV-14	13.49	5	5	1	(null)
4	3	The Binding of Isaac: Afterbirth	30-OCT-15	9.89	5	5	1	2
5	4	The Binding of Isaac: Afterbirth+	03-JAN-17	8.99	5	5	1	3
6	5	The Binding of Isaac: Repentance	31-MAR-21	13.13	5	5	1	4
7	6	Pokemon Sword and Shield	15-NOV-21	27	3	3	2	(null)

Excepție:



```
BEGIN
    discount_all_independent(6);
END;
/
```

Dbms Output ×

Buffer Size: 20000

Local ×

```
INVALID DISCOUNT VALUE
```

8. Funcție care să definească cel puțin două excepții și care să folosească cel puțin 3 tabele într-o expresie SQL

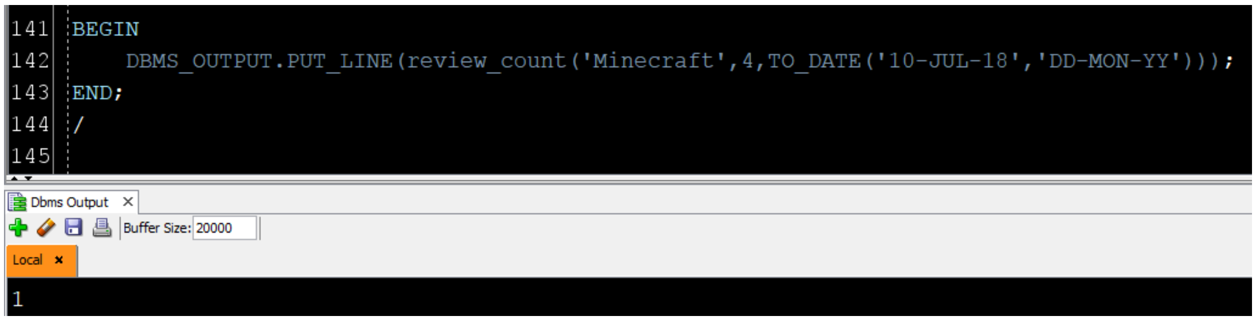
Enunț: Să se returneze numărul de recenzii cu scor mai mare sau egal cu o valoare data, pentru un anumit produs, făcute după o anumită dată doar de către utilizatorii care dețin produsul.

Am definit: INVALID_THRESHOLD și INVALID_DATE

Am folosit tabelele: reviews, product_purchases și purchases.

Explicație: Dacă pragul de scor introdus nu este între 0-5 arunc excepția INVALID_THRESHOLD, deoarece rating ia valori doar între 0 și 5. Extragem product_id și release_date pentru produsul introdus; poate apărea NO_DATA_FOUND. Dacă pragul de dată introdus nu este între release_date și sysdate, arunc excepția INVALID_DATE. Apoi printr-o singură expresie SQL număr recenziile care satisfac condițiile. Subcererea returnează toți utilizatorii care dețin produsul(sub formă de tuplu utilizator-produs).

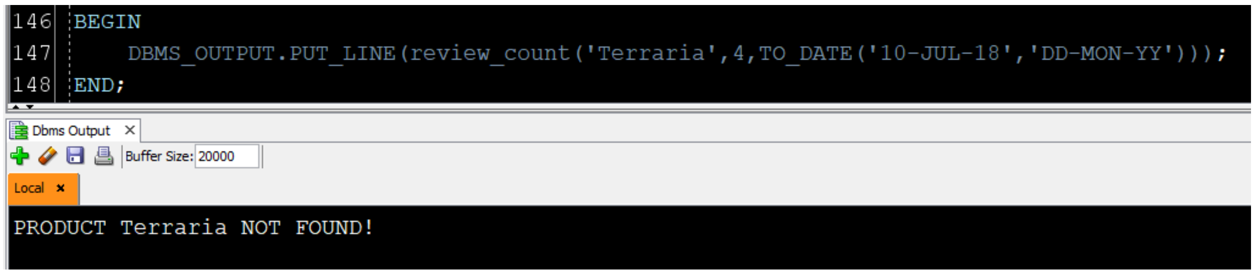
```
FUNCTION review_count(p_product IN products.product_title%TYPE,
                                        p_threshold IN reviews.rating%TYPE,
                                        p_date_since IN reviews.review_date%TYPE)
                                        RETURN NUMBER IS
        v_count NUMBER;
        v_id products.product_id%TYPE;
        v_release products.release_date%TYPE;
        INVALID_THRESHOLD EXCEPTION;
        INVALID_DATE EXCEPTION;
    BEGIN
        IF p_threshold NOT BETWEEN 0 AND 5 THEN RAISE INVALID_THRESHOLD;
        END IF;

        SELECT product_id,release_date
        INTO v_id,v_release
        FROM products
        WHERE lower(product_title) = lower(p_product);

        IF p_date_since NOT BETWEEN v_release AND sysdate THEN RAISE INVALID_DATE;
        END IF;

        SELECT count(*)
        INTO v_count                       --3 tables
        FROM reviews
        WHERE (account_id,product_id) IN (SELECT receiver_id, product_id
                                          FROM product_purchases JOIN  purchases USING
(client_id,purchase_date)
                                          WHERE product_id = v_id)
               AND rating >= p_threshold
               AND review_date >= p_date_since;
        RETURN v_count;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN DBMS_OUTPUT.PUT_LINE('PRODUCT '||p_product||' NOT FOUND!');
                                RAISE NO_DATA_FOUND;
        WHEN INVALID_THRESHOLD THEN DBMS_OUTPUT.PUT_LINE('INVALID THRESHOLD! SHOULD BE BETWEEN 0 and
5!');
        WHEN INVALID_DATE THEN DBMS_OUTPUT.PUT_LINE('INVALID DATE! SHOULD BE BETWEEN PRODUCT DATE AND
PRESENT');
    END review_count;
```



```
BEGIN
    DBMS_OUTPUT.PUT_LINE(review_count('Minecraft',4,TO_DATE('10-JUL-18','DD-MON-YY')));
END;
/
```

Buffer Size: 20000

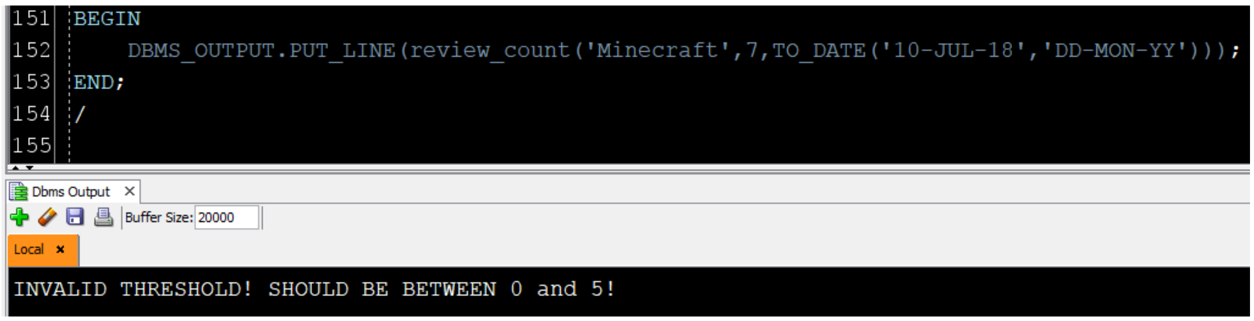
```
1
```



```
BEGIN
    DBMS_OUTPUT.PUT_LINE(review_count('Terraria',4,TO_DATE('10-JUL-18','DD-MON-YY')));
END;
```

Buffer Size: 20000

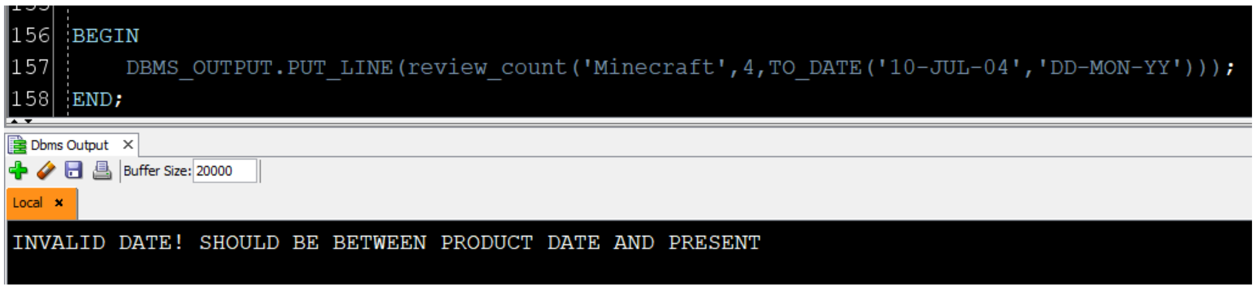
```
PRODUCT Terraria NOT FOUND!
```



```
BEGIN
    DBMS_OUTPUT.PUT_LINE(review_count('Minecraft',7,TO_DATE('10-JUL-18','DD-MON-YY')));
END;
/
```

Buffer Size: 20000

```
INVALID THRESHOLD! SHOULD BE BETWEEN 0 and 5!
```



```
BEGIN
    DBMS_OUTPUT.PUT_LINE(review_count('Minecraft',4,TO_DATE('10-JUL-04','DD-MON-YY')));
END;
```

Buffer Size: 20000

```
INVALID DATE! SHOULD BE BETWEEN PRODUCT DATE AND PRESENT
```

9. Procedură care să folosească minim 5 tabele într-o singură expresie SQL.

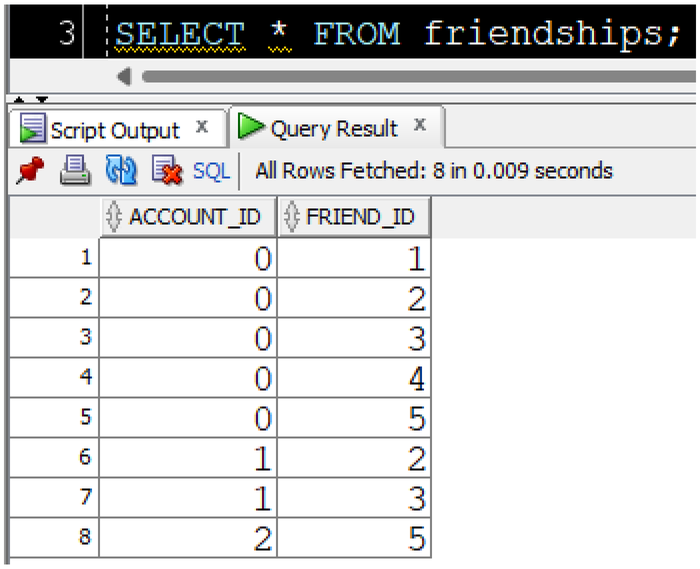
Enunț: Pentru un utilizator dat(display name), sa se șteargă toate prieteniile cu utilizatori care nu au niciun produs în comun pe o platforma comună.

Am folosit tabelele: friendships, accounts, purchases, product_purchases și product_availability.

Explicație: Extragem account_id pentru display_name introdus. Poate apărea atât NO_DATA_FOUND cât și TOO_MANY_ROWS. Extragem toți prietenii utilizatorului într-o colecție(relația este simetrică deci utilizatorul dat poate fi atât account_id cât și friend_id din friendships, deci fac union. Apoi extrag utilizatorii care au produse comune pe aceleași platforme cu utilizatorul dat: prin niște joinuri determin toate tuplurile produs-platformă pe care le dețin prietenii, și apoi le filtrez să corespundă cu cele ale utilizatorului, determinate prin subcerere. În final, făcând MULTISET EXCEPT între cele două colecții obțin prietenii care nu au nimic în comun cu userul.

```
PROCEDURE delete_uncommon_friends(p_user IN accounts.display_name%TYPE) IS
        --procedure could be simplified by declaring this collection at schema level
        --thus enabling me to use it in SQL expressions... I chose not to do that however
        TYPE account_table IS TABLE OF accounts.account_id%TYPE;
        v_user accounts.account_id%TYPE;
        v_all_friends account_table := account_table();
        v_common_friends account_table := account_table();
        v_uncommon_friends account_table := account_table();
    BEGIN
        SELECT account_id
        INTO v_user
        FROM accounts
        WHERE lower(display_name) = lower(p_user);

        WITH friends AS
            (SELECT a.account_id
            FROM friendships   f
            JOIN accounts a ON (f.friend_id = a.account_id)
            WHERE f.account_id = v_user
            UNION
            SELECT account_id
            FROM friendships
            JOIN accounts USING (account_id)
            WHERE friend_id = v_user)
        SELECT account_id
        BULK COLLECT INTO v_all_friends
        FROM friends;

        WITH friends AS
            (SELECT a.account_id
            FROM friendships   f
            JOIN accounts a ON (f.friend_id = a.account_id)
            WHERE f.account_id = v_user
            UNION
            SELECT account_id
            FROM friendships
            JOIN accounts USING (account_id)
            WHERE friend_id = v_user)
        SELECT DISTINCT a.account_id
        BULK COLLECT INTO v_common_friends
        FROM friends a JOIN purchases p ON (a.account_id = p.receiver_id)      --5 tables
             JOIN product_purchases pp ON (pp.client_id = p.client_id
                                  AND pp.purchase_date = p.purchase_date)
             JOIN product_availability pa ON (pa.product_id = pp.product_id)
        WHERE(pa.product_id,pa.platform_id) IN (SELECT pa.product_id, pa.platform_id
                                                FROM purchases p
                                                JOIN product_purchases pp ON (pp.client_id = p.client_id
                                                                          AND pp.purchase_date =
p.purchase_date)
                                                JOIN product_availability pa ON (pa.product_id =
pp.product_id)
                                                WHERE receiver_id = v_user);

         v_uncommon_friends := v_all_friends MULTISET EXCEPT v_common_friends;

         FORALL i IN v_uncommon_friends.FIRST..v_uncommon_friends.LAST
         DELETE FROM friendships
         WHERE (account_id = v_user AND friend_id = v_uncommon_friends(i))
            OR (friend_id = v_user AND account_id = v_uncommon_friends(i));

    EXCEPTION
        WHEN NO_DATA_FOUND THEN DBMS_OUTPUT.PUT_LINE('USER WITH DISPLAY NAME '|| p_user || ' NOT
FOUND');
                                RAISE NO_DATA_FOUND;
        WHEN TOO_MANY_ROWS THEN DBMS_OUTPUT.PUT_LINE('MORE THAN ONE USER WITH DISPLAY NAME ' || p_user);
                                RAISE TOO_MANY_ROWS;
    END delete_uncommon_friends;
```

Înainte:



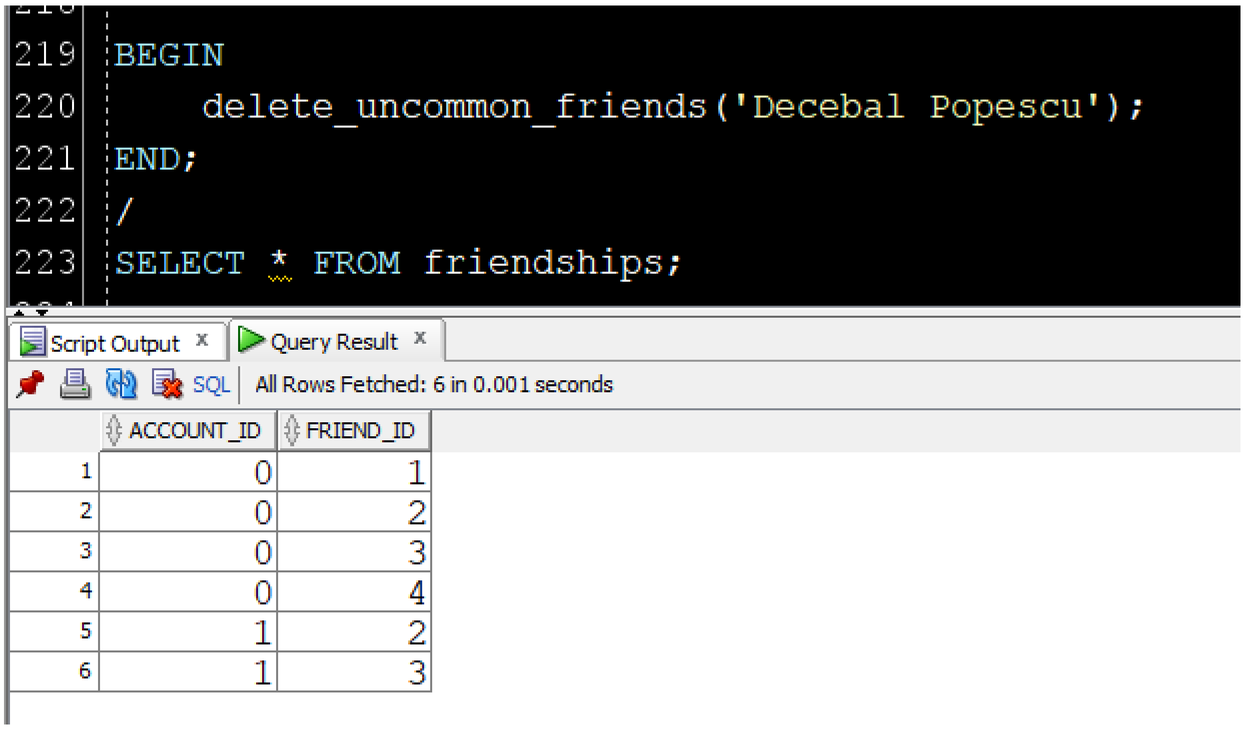
```
SELECT * FROM friendships;
```

Script Output | Query Result

All Rows Fetched: 8 in 0.009 seconds

	ACCOUNT_ID	FRIEND_ID
1	0	1
2	0	2
3	0	3
4	0	4
5	0	5
6	1	2
7	1	3
8	2	5

După:



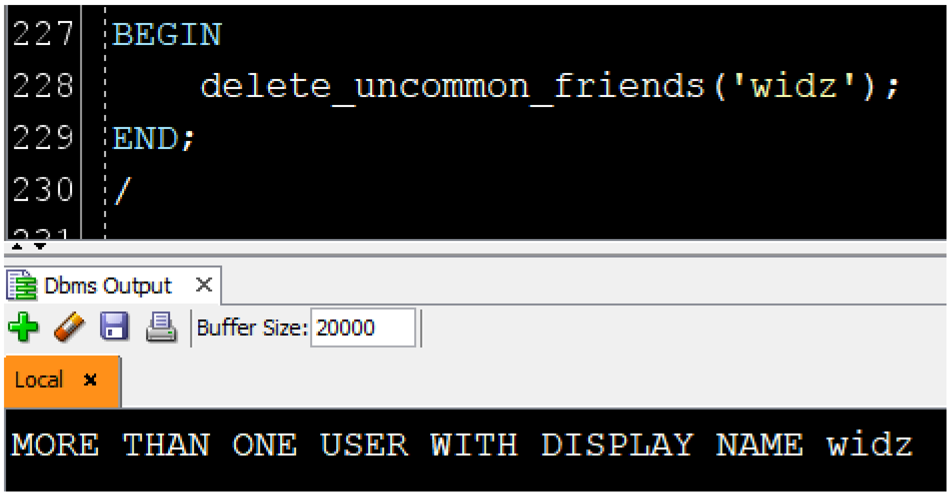
```
BEGIN
    delete_uncommon_friends('Decebal Popescu');
END;
/
SELECT * FROM friendships;
```

Script Output | Query Result

All Rows Fetched: 6 in 0.001 seconds

	ACCOUNT_ID	FRIEND_ID
1	0	1
2	0	2
3	0	3
4	0	4
5	1	2
6	1	3

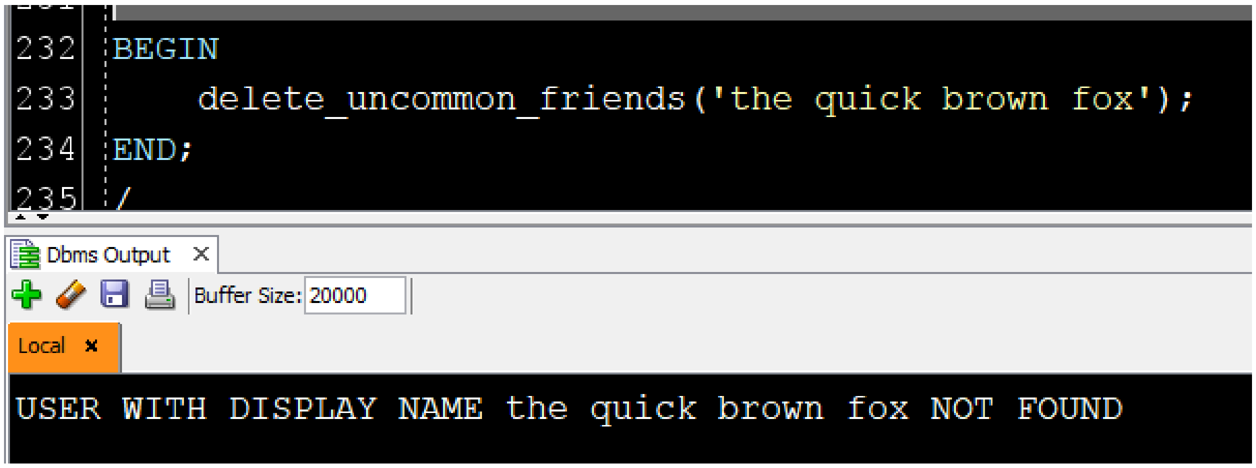
Excepții:



```
BEGIN
    delete_uncommon_friends('widz');
END;
/
```

Dbms Output — Buffer Size: 20000 — Local

```
MORE THAN ONE USER WITH DISPLAY NAME widz
```



```
BEGIN
    delete_uncommon_friends('the quick brown fox');
END;
/
```

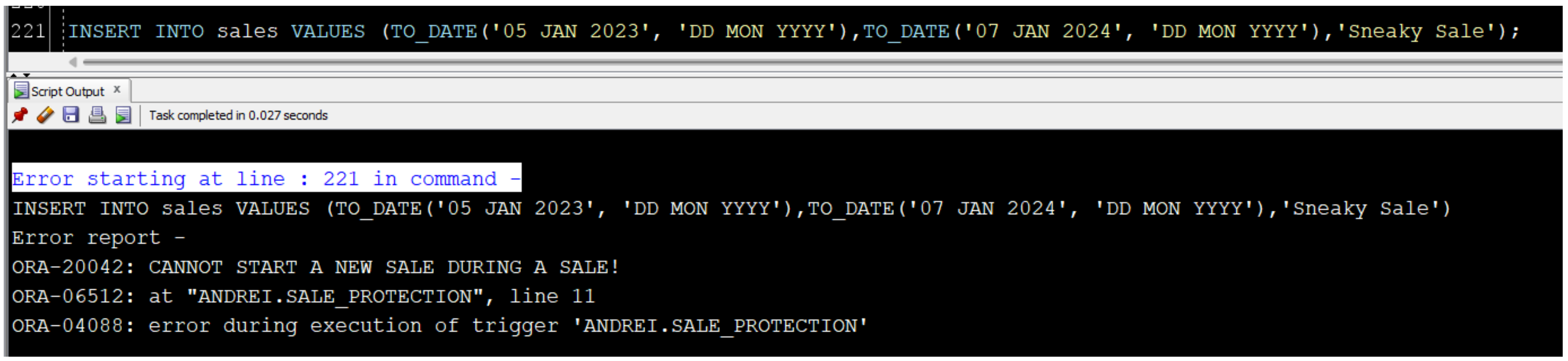
Dbms Output — Buffer Size: 20000 — Local

```
USER WITH DISPLAY NAME the quick brown fox NOT FOUND
```

10. Trigger LMD comandă

Dacă un sezon de reducere este în desfășurare, nu este permisă inserarea altuia nou.

```
CREATE OR REPLACE TRIGGER sale_protection
    BEFORE INSERT ON sales
    DECLARE
        v_start DATE;
        v_end DATE;
    BEGIN
        SELECT start_date, end_date
        INTO v_start,v_end
        FROM sales
        WHERE end_date = (SELECT max(end_date)
                          FROM sales);

        IF sysdate BETWEEN v_start AND v_end THEN RAISE_APPLICATION_ERROR(-20042,'CANNOT START A NEW
SALE DURING A SALE!');
        END IF;
    EXCEPTION
          WHEN NO_DATA_FOUND THEN NULL;
END;
/
```

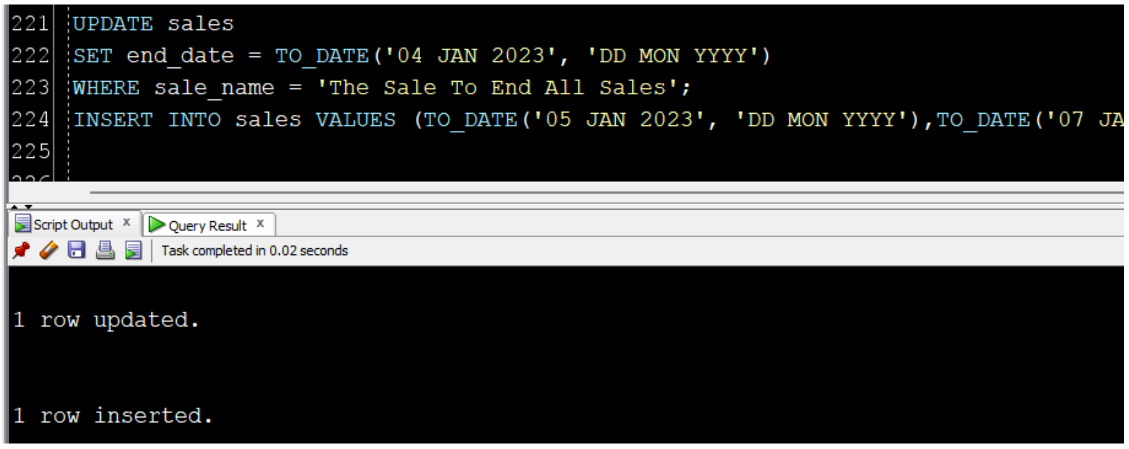


```
INSERT INTO sales VALUES (TO_DATE('05 JAN 2023', 'DD MON YYYY'),TO_DATE('07 JAN 2024', 'DD MON YYYY'),'Sneaky Sale');
```

Script Output
Task completed in 0.027 seconds

```
Error starting at line : 221 in command -
INSERT INTO sales VALUES (TO_DATE('05 JAN 2023', 'DD MON YYYY'),TO_DATE('07 JAN 2024', 'DD MON YYYY'),'Sneaky Sale')
Error report -
ORA-20042: CANNOT START A NEW SALE DURING A SALE!
ORA-06512: at "ANDREI.SALE_PROTECTION", line 11
ORA-04088: error during execution of trigger 'ANDREI.SALE_PROTECTION'
```

Dacă modificăm astfel încât să nu mai fim într-un sezon de reduceri:



```
UPDATE sales
SET end_date = TO_DATE('04 JAN 2023', 'DD MON YYYY')
WHERE sale_name = 'The Sale To End All Sales';
INSERT INTO sales VALUES (TO_DATE('05 JAN 2023', 'DD MON YYYY'),TO_DATE('07 JA
```

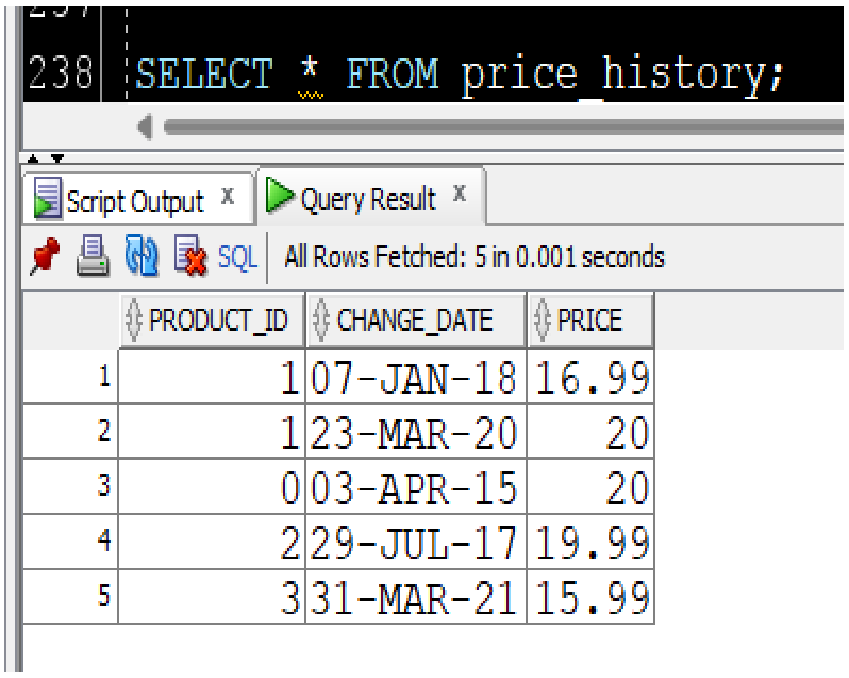
Script Output | Query Result
Task completed in 0.02 seconds

```
1 row updated.


1 row inserted.
```

11. Trigger LMD linie

Când se modifică prețul unui produs, să se insereze automat vechiul preț în price_history.

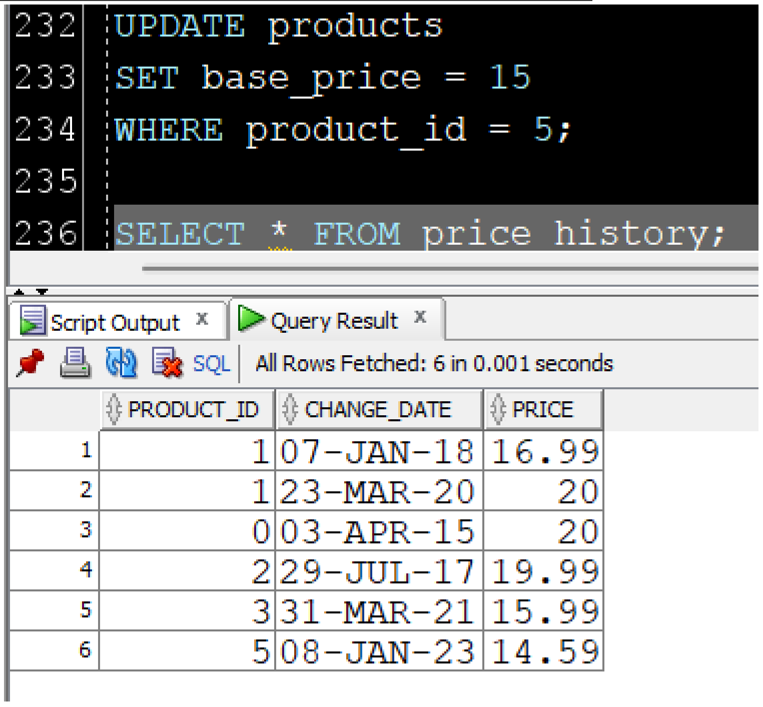
```
CREATE OR REPLACE TRIGGER price_change
    AFTER UPDATE OF base_price ON products
    FOR EACH ROW
    BEGIN
        INSERT INTO price_history
        VALUES (:old.product_id, sysdate, :old.base_price);
END;
/
```



```
SELECT * FROM price_history;
```

All Rows Fetched: 5 in 0.001 seconds

	PRODUCT_ID	CHANGE_DATE	PRICE
1	1	07-JAN-18	16.99
2	1	23-MAR-20	20
3	0	03-APR-15	20
4	2	29-JUL-17	19.99
5	3	31-MAR-21	15.99



```
UPDATE products
SET base_price = 15
WHERE product_id = 5;

SELECT * FROM price_history;
```

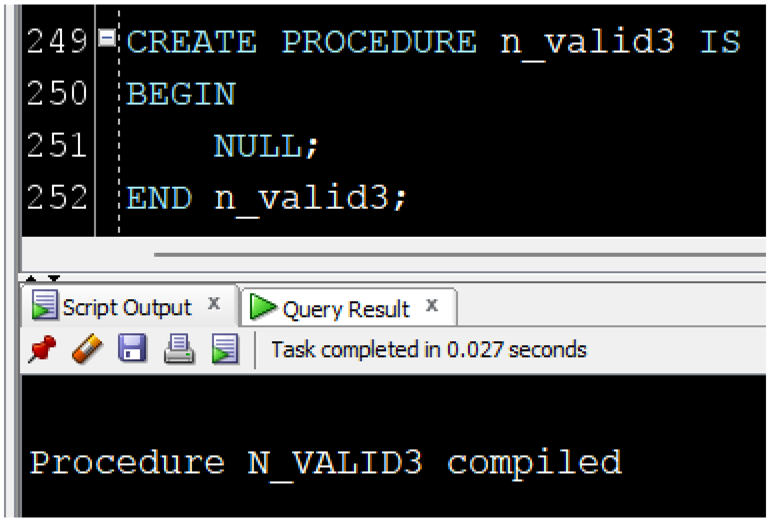
All Rows Fetched: 6 in 0.001 seconds

	PRODUCT_ID	CHANGE_DATE	PRICE
1	1	07-JAN-18	16.99
2	1	23-MAR-20	20
3	0	03-APR-15	20
4	2	29-JUL-17	19.99
5	3	31-MAR-21	15.99
6	5	08-JAN-23	14.59

12. Trigger LDD

Să se valideze numele oricărui obiect din schemă: minim 2 litere, doar caractere alfanumerice și _, nu poate începe cu sql sau dba.

```
CREATE OR REPLACE TRIGGER name_validation
    BEFORE CREATE ON SCHEMA
    BEGIN

    IF regexp_like(ora_dict_obj_name,'^.{0,2}$|^(sql|dba)|[^a-z^0-9_]','i') THEN
RAISE_APPLICATION_ERROR(-20043,'INVALID OBJECT NAME!');
    END IF;
END;
/
```

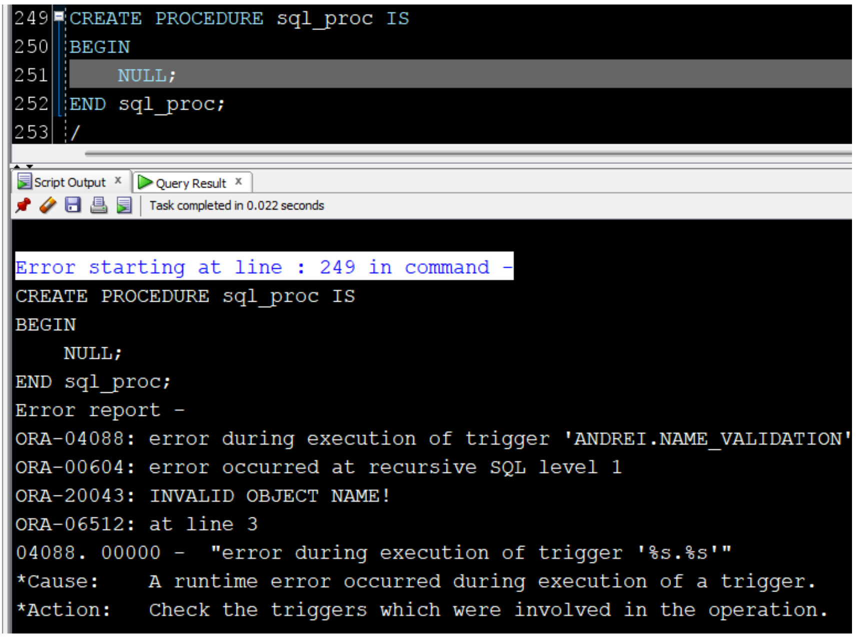


```
CREATE PROCEDURE n_valid3 IS
BEGIN
    NULL;
END n_valid3;
```

Script Output | Query Result

Task completed in 0.027 seconds

```
Procedure N_VALID3 compiled
```



```
CREATE PROCEDURE sql_proc IS
BEGIN
    NULL;
END sql_proc;
/
```

Script Output | Query Result

Task completed in 0.022 seconds

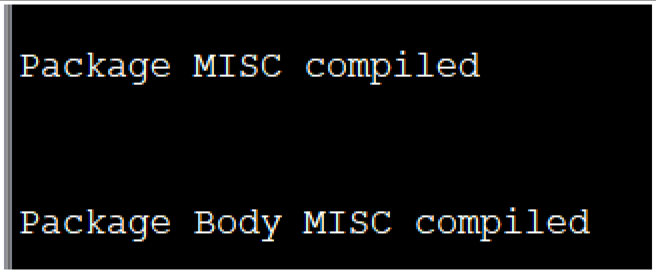
```
Error starting at line : 249 in command -
CREATE PROCEDURE sql_proc IS
BEGIN
    NULL;
END sql_proc;
Error report -
ORA-04088: error during execution of trigger 'ANDREI.NAME_VALIDATION'
ORA-00604: error occurred at recursive SQL level 1
ORA-20043: INVALID OBJECT NAME!
ORA-06512: at line 3
04088. 00000 -  "error during execution of trigger '%s.%s'"
*Cause:    A runtime error occurred during execution of a trigger.
*Action:   Check the triggers which were involved in the operation.
```

13. Pachet cu toate obiectele

```
CREATE OR REPLACE PACKAGE misc AS
    BAD_DISCOUNT EXCEPTION;
    INVALID_THRESHOLD EXCEPTION;
    INVALID_DATE EXCEPTION;

    FUNCTION average_diff(p_platform_name IN platforms.platform_name%TYPE) RETURN NUMBER;
    PROCEDURE discount_all_independent(p_discount IN products.base_price%TYPE);
    FUNCTION review_count(p_product IN products.product_title%TYPE,
                                        p_threshold IN reviews.rating%TYPE,
                                        p_date_since IN reviews.review_date%TYPE)
                                        RETURN NUMBER;
    PROCEDURE delete_uncommon_friends(p_user IN accounts.display_name%TYPE);
END misc;
/

CREATE OR REPLACE PACKAGE BODY misc AS

--Pentru o platforma dată, să se returneze diferența medie dintre prețul fiecărui produs
--și prețul celui mai scump produs din franciza respectivă.
--(dacă nu face parte din vreo franciza, se considera pretul maxim 0.)
    FUNCTION average_diff(p_platform_name IN platforms.platform_name%TYPE) RETURN NUMBER IS
        TYPE fran_prod_map IS TABLE OF products.base_price%TYPE INDEX BY PLS_INTEGER;
        TYPE prod_rec IS RECORD
            (prod_id products.product_id%TYPE,
            fran_id products.franchise_id%TYPE,
            price products.base_price%TYPE);
        TYPE prod_tab IS TABLE OF prod_rec;
        v_products prod_tab := prod_tab();
        v_max_prod fran_prod_map;
        v_average NUMBER := 0;
        v_plat_id platforms.platform_id%TYPE;
    BEGIN
        SELECT platform_id
        INTO v_plat_id
        FROM platforms
        WHERE platform_name = p_platform_name;

        SELECT product_id,franchise_id,base_price
        BULK COLLECT INTO v_products
        FROM product_availability JOIN products USING (product_id)
        WHERE platform_id = v_plat_id;

        FOR i IN v_products.FIRST..v_products.LAST LOOP --we know for sure it's dense
           IF v_products(i).fran_id IS NULL
                THEN  v_average := v_average + v_products(i).price;
                    CONTINUE;
            END IF;

            IF NOT v_max_prod.EXISTS(v_products(i).fran_id) --if we've already computed the maximum for
this franchise
                THEN SELECT MAX(base_price)
                     INTO v_max_prod(v_products(i).fran_id)
                     FROM products
                     WHERE franchise_id = v_products(i).fran_id;
            END IF;
            v_average := v_average + v_max_prod(v_products(i).fran_id) - v_products(i).price;
        END LOOP;

        RETURN (v_average/v_products.COUNT);
    EXCEPTION
        WHEN NO_DATA_FOUND THEN DBMS_OUTPUT.PUT_LINE('PLATFORM ' || p_platform_name ||' NOT FOUND!');
```

```
                                RAISE NO_DATA_FOUND;
        WHEN VALUE_ERROR THEN DBMS_OUTPUT.PUT_LINE('THERE ARE NO PRODUCTS FOR PLATFORM ' ||
p_platform_name);
    END average_diff;

--Să se reducă cu un procent dat prețul tuturor produselor dezvoltate de către studiouri independente
--(care nu au alt studio parinte).
    PROCEDURE discount_all_independent(p_discount IN products.base_price%TYPE) IS
        CURSOR products_by(studio_id studios.studio_id%TYPE) IS --parameter cursor
            SELECT product_id
            FROM products
            WHERE developer_id = studio_id
            FOR UPDATE OF base_price;
        BAD_DISCOUNT EXCEPTION;
    BEGIN
        IF p_discount NOT BETWEEN 0 AND 1 THEN RAISE BAD_DISCOUNT;
        END IF;
        FOR studio IN (SELECT studio_id     -- subquery cursor
                       FROM studios
                       WHERE parent_id IS NULL) LOOP
            FOR product IN products_by(studio.studio_id) LOOP
                UPDATE products
                SET base_price = base_price * (1 - p_discount)
                WHERE CURRENT OF products_by;
            END LOOP;
        END LOOP;
    EXCEPTION
        WHEN BAD_DISCOUNT THEN DBMS_OUTPUT.PUT_LINE('INVALID DISCOUNT VALUE');
    END discount_all_independent;

--Să se returneze numărul de recenzii cu scor mai mare sau egal cu o valoare data,
--pentru un anumit produs, făcute după o anumită dată
--doar de către utilizatorii care dețin produsul.
    FUNCTION review_count(p_product IN products.product_title%TYPE,
                                        p_threshold IN reviews.rating%TYPE,
                                        p_date_since IN reviews.review_date%TYPE)
                                        RETURN NUMBER IS
        v_count NUMBER;
        v_id products.product_id%TYPE;
        v_release products.release_date%TYPE;
    BEGIN
        IF p_threshold NOT BETWEEN 0 AND 5 THEN RAISE INVALID_THRESHOLD;
        END IF;

        SELECT product_id,release_date
        INTO v_id,v_release
        FROM products
        WHERE lower(product_title) = lower(p_product);

        IF p_date_since NOT BETWEEN v_release AND sysdate THEN RAISE INVALID_DATE;
        END IF;

        SELECT count(*)
        INTO v_count                       --3 tables
        FROM reviews
        WHERE (account_id,product_id) IN (SELECT receiver_id, product_id
                                          FROM product_purchases JOIN  purchases USING
(client_id,purchase_date)
                                          WHERE product_id = v_id)
               AND rating >= p_threshold
               AND review_date >= p_date_since;
        RETURN v_count;
    EXCEPTION
```

```
        WHEN NO_DATA_FOUND THEN DBMS_OUTPUT.PUT_LINE('PRODUCT '||p_product||' NOT FOUND!');
                                RAISE NO_DATA_FOUND;
        WHEN INVALID_THRESHOLD THEN DBMS_OUTPUT.PUT_LINE('INVALID THRESHOLD! SHOULD BE BETWEEN 0 and
5!');
        WHEN INVALID_DATE THEN DBMS_OUTPUT.PUT_LINE('INVALID DATE! SHOULD BE BETWEEN PRODUCT'S DATE AND
PRESENT');
    END review_count;

--Pentru un utilizator dat(display name), sa se șteargă toate prieteniile
--cu utilizatori care nu au niciun produs în comun pe o platforma comună.
    PROCEDURE delete_uncommon_friends(p_user IN accounts.display_name%TYPE) IS
        --procedure could be simplified by declaring this collection at schema level
        --thus enabling me to use it in SQL expressions... I chose not to do that however
        TYPE account_table IS TABLE OF accounts.account_id%TYPE;
        v_user accounts.account_id%TYPE;
        v_all_friends account_table := account_table();
        v_common_friends account_table := account_table();
        v_uncommon_friends account_table := account_table();
    BEGIN
        SELECT account_id
        INTO v_user
        FROM accounts
        WHERE lower(display_name) = lower(p_user);

        WITH friends AS
            (SELECT a.account_id
            FROM friendships   f
            JOIN accounts a ON (f.friend_id = a.account_id)
            WHERE f.account_id = v_user
            UNION
            SELECT account_id
            FROM friendships
            JOIN accounts USING (account_id)
            WHERE friend_id = v_user)
        SELECT account_id
        BULK COLLECT INTO v_all_friends
        FROM friends;

        WITH friends AS
            (SELECT a.account_id
            FROM friendships   f
            JOIN accounts a ON (f.friend_id = a.account_id)
            WHERE f.account_id = v_user
            UNION
            SELECT account_id
            FROM friendships
            JOIN accounts USING (account_id)
            WHERE friend_id = v_user)
        SELECT DISTINCT a.account_id
        BULK COLLECT INTO v_common_friends
        FROM friends a JOIN purchases p ON (a.account_id = p.receiver_id)      --5 tables
             JOIN product_purchases pp ON (pp.client_id = p.client_id
                                     AND pp.purchase_date = p.purchase_date)
             JOIN product_availability pa ON (pa.product_id = pp.product_id)
        WHERE(pa.product_id,pa.platform_id) IN (SELECT pa.product_id, pa.platform_id
                                                FROM purchases p
                                                JOIN product_purchases pp ON (pp.client_id = p.client_id
                                                                          AND pp.purchase_date =
p.purchase_date)
                                                JOIN product_availability pa ON (pa.product_id =
pp.product_id)
                                                WHERE receiver_id = v_user);
        v_uncommon_friends := v_all_friends MULTISET EXCEPT v_common_friends;

        FORALL i IN v_uncommon_friends.FIRST..v_uncommon_friends.LAST
        DELETE FROM friendships
        WHERE (account_id = v_user AND friend_id = v_uncommon_friends(i))
           OR (friend_id = v_user AND account_id = v_uncommon_friends(i));

    EXCEPTION
        WHEN NO_DATA_FOUND THEN DBMS_OUTPUT.PUT_LINE('USER WITH DISPLAY NAME '|| p_user || ' NOT
FOUND');
                                RAISE NO_DATA_FOUND;
        WHEN TOO_MANY_ROWS THEN DBMS_OUTPUT.PUT_LINE('MORE THAN ONE USER WITH DISPLAY NAME ' || p_user);
                                RAISE TOO_MANY_ROWS;
    END delete_uncommon_friends;

END misc;
/
```



```
Package MISC compiled


Package Body MISC compiled
```

14. Pachet pentru un flux

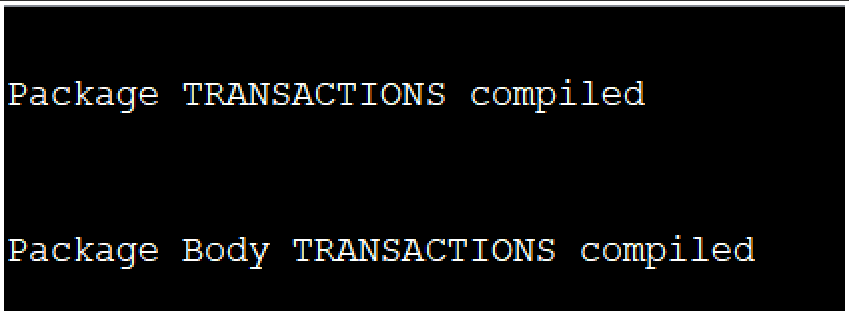
Am creat un pachet pentru a facilita tranzacțiile(monetare, nu de baze de date). Funcționalitățile sunt:

- Colecție pentru produse (id + titlu)
- Record pentru o achiziție (toate informațiile + listă de produse)
- Funcție care returnează prețul unui anumit produs la o anumită dată, inclusiv cu reduceri.
- Funcție care returnează recordul pentru o achiziție(identificată prin client și dată)
- Funcție care returnează totalul unei achiziții
- Funcție care returnează soldul unui utilizator
- Procedură care adaugă un nou produs într-un “coș”
- Procedură care elimină un produs din “coș”
- Procedură care efectuează o achiziție
- Procedură care efectuează un depozit

```
CREATE OR REPLACE PACKAGE transactions AS
    INVALID_SUM EXCEPTION;
    INSUFFICIENT_FUNDS EXCEPTION;
    INVALID_DATE EXCEPTION;
    TYPE product IS RECORD(
        id products.product_id%TYPE,
        title products.product_title%TYPE);
    TYPE product_list IS TABLE OF product;
    TYPE purchase IS RECORD(
        client_id purchases.client_id%TYPE,
        receiver_id purchases.receiver_id%TYPE,
        purchase_date purchases.purchase_date%TYPE,
        products product_list);
    FUNCTION getPrice(p_product IN products.product_title%TYPE, p_date IN DATE) RETURN products.base_price%TYPE;
    FUNCTION getPurchase(p_client IN accounts.username%TYPE, p_date IN DATE) RETURN purchase;
    FUNCTION getPurchaseTotal(p_purchase IN purchase) RETURN products.base_price%TYPE;
    FUNCTION getUserBalance(p_user IN accounts.username%TYPE) RETURN deposits.deposit_sum%TYPE;
    PROCEDURE addToCart(p_product IN products.product_title%TYPE, p_cart IN OUT product_list);
    PROCEDURE removeFromCart(p_product IN products.product_title%TYPE, p_cart IN OUT product_list);
    PROCEDURE makePurchase(p_client IN accounts.username%TYPE, p_cart IN product_list, p_receiver IN
accounts.username%TYPE := NULL);
    PROCEDURE makeDeposit(p_user IN accounts.username%TYPE, p_sum IN deposits.deposit_sum%TYPE);
END transactions;
/

CREATE OR REPLACE PACKAGE BODY transactions AS
    FUNCTION getPrice(p_product IN products.product_title%TYPE, p_date IN DATE) RETURN products.base_price%TYPE IS
    v_prod_id products.product_id%TYPE;
    v_price products.base_price%TYPE;
    v_discount discount_history.discount%TYPE;
    v_release DATE;
    BEGIN
        SELECT product_id,base_price,release_date
        INTO v_prod_id,v_price,v_release
        FROM products
        WHERE lower(product_title) = lower(p_product);

        IF p_date NOT BETWEEN v_release AND sysdate THEN RAISE INVALID_DATE;
        END IF;

        BEGIN
            SELECT price
            INTO v_price
            FROM price_history
            WHERE product_id = v_prod_id AND change_date =
                            (SELECT min(change_date)
                            FROM price_history
                            WHERE product_id = v_prod_id AND p_date < change_date);
        EXCEPTION
            WHEN NO_DATA_FOUND THEN NULL;
        END;

        BEGIN
            SELECT discount
            INTO v_discount
            FROM discount_history JOIN SALES ON (sale_date = start_date)
            WHERE product_id = v_prod_id
                AND p_date BETWEEN start_date AND end_date;
        EXCEPTION
            WHEN NO_DATA_FOUND THEN v_discount := 0;
        END;



        RETURN v_price *(1 - v_discount);
    EXCEPTION
        WHEN NO_DATA_FOUND THEN DBMS_OUTPUT.PUT_LINE('PRODUCT '||p_product ||' NOT FOUND');
                                RAISE NO_DATA_FOUND;
        WHEN INVALID_DATE THEN DBMS_OUTPUT.PUT_LINE('DATE IS INVALID! HAS TO BE BETWEEN PRODUCT DATE AND PRESENT!');
                                RAISE INVALID_DATE;
    END getPrice;
    FUNCTION getPurchase(p_client IN accounts.username%TYPE, p_date IN DATE) RETURN purchase IS
        v_purchase purchase;
    BEGIN

        SELECT client_id, purchase_date, receiver_id
        INTO v_purchase.client_id, v_purchase.purchase_date, v_purchase.receiver_id
        FROM purchases JOIN accounts ON (account_id = client_id)
        WHERE lower(username) = lower(p_client) AND purchase_date = p_date;

        v_purchase.products := product_list();

        SELECT product_id, product_title
        BULK COLLECT INTO v_purchase.products
        FROM product_purchases JOIN products USING(product_id)
        WHERE client_id = v_purchase.client_id AND purchase_date = v_purchase.purchase_date;

        return v_purchase;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN DBMS_OUTPUT.PUT_LINE('THERE IS NO PURCHASE DONE BY '||p_client||' AT DATE
'||to_char(p_date,'DD/MM/YYYY'));
                                RAISE NO_DATA_FOUND;
    END getPurchase;

    FUNCTION getPurchaseTotal(p_purchase IN purchase) RETURN products.base_price%TYPE IS
        v_total products.base_price%TYPE := 0;
        v_index PLS_INTEGER;
    BEGIN
         v_index := p_purchase.products.first;
         WHILE v_index IS NOT NULL LOOP
            v_total := v_total + getPrice(p_purchase.products(v_index).title,p_purchase.purchase_date);
            v_index := p_purchase.products.next(v_index);
         END LOOP;

         RETURN v_total;
    END getPurchaseTotal;

    FUNCTION getUserBalance(p_user IN accounts.username%TYPE) RETURN deposits.deposit_sum%TYPE IS
        v_balance deposits.deposit_sum%TYPE :=0;
        v_spent deposits.deposit_sum%TYPE :=0;
        v_user_id accounts.account_id%TYPE;
    BEGIN

        SELECT account_id,sum(deposit_sum)
        INTO v_user_id,v_balance
        FROM deposits JOIN accounts ON (client_id = account_id)
        WHERE lower(username) = lower(p_user)
        GROUP BY account_id;

        FOR purchase IN (SELECT purchase_date
                         FROM purchases
                         WHERE client_id = v_user_id) LOOP
            v_spent := v_spent + getPurchaseTotal(getPurchase(p_user,purchase.purchase_date));
        END LOOP;
        RETURN v_balance - v_spent;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN DBMS_OUTPUT.PUT_LINE('USER ' || p_user || ' NOT FOUND!');
                                RAISE NO_DATA_FOUND;
    END getUserBalance;

    PROCEDURE addToCart(p_product IN products.product_title%TYPE, p_cart IN OUT product_list) IS
        v_exists BOOLEAN := FALSE;
        v_index PLS_INTEGER;
        v_prod product;
    BEGIN
        SELECT product_id,product_title
        INTO v_prod
        FROM products
        WHERE lower(product_title) = lower(p_product);

        v_index := p_cart.first;
        WHILE v_index IS NOT NULL LOOP
            IF lower(p_cart(v_index).title) = lower(p_product)
```

```
                THEN v_exists := TRUE;
                EXIT;
            END IF;
            v_index := p_cart.next(v_index);
        END LOOP;

        IF NOT v_exists
            THEN p_cart.extend();
            p_cart(p_cart.last) := v_prod;
        END IF;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN DBMS_OUTPUT.PUT_LINE('PRODUCT '||p_product ||' NOT FOUND');
                                RAISE NO_DATA_FOUND;
    END addToCart;

    PROCEDURE removeFromCart(p_product IN products.product_title%TYPE, p_cart IN OUT product_list) IS
        v_index PLS_INTEGER;
    BEGIN
        v_index := p_cart.first;
        WHILE v_index IS NOT NULL LOOP
            IF lower(p_cart(v_index).title) = lower(p_product)
                THEN p_cart.delete(v_index);
                EXIT;
            END IF;
            v_index := p_cart.next(v_index);
        END LOOP;
    END;

    PROCEDURE makePurchase(p_client IN accounts.username%TYPE, p_cart IN product_list, p_receiver IN
accounts.username%TYPE := NULL) IS
        INSUFFICIENT_FUNDS EXCEPTION;
        v_purchase purchase;
    BEGIN

        SELECT account_id
        INTO v_purchase.client_id
        FROM accounts
        WHERE lower(username) = lower(p_client);

        IF p_receiver IS NULL
            THEN v_purchase.receiver_id := v_purchase.client_id;
        ELSE
            SELECT account_id
            INTO v_purchase.receiver_id
            FROM accounts
            WHERE lower(username) = lower(p_receiver);
        END IF;

        v_purchase.products := p_cart;
        v_purchase.purchase_date := sysdate;

        IF getPurchaseTotal(v_purchase) > getUserBalance(p_client)
            THEN RAISE INSUFFICIENT_FUNDS;
        END IF;

        INSERT INTO purchases
        VALUES (v_purchase.client_id,v_purchase.purchase_date,v_purchase.receiver_id);

        FORALL i IN INDICES OF v_purchase.products
            INSERT INTO product_purchases
            VALUES (v_purchase.products(i).id,v_purchase.client_id,v_purchase.purchase_date);

    EXCEPTION
        WHEN NO_DATA_FOUND THEN DBMS_OUTPUT.PUT_LINE('USER NOT FOUND!');
                                RAISE NO_DATA_FOUND;
        WHEN INSUFFICIENT_FUNDS THEN DBMS_OUTPUT.PUT_LINE('USER HAS INSUFFICIENT FUNDS!');
                                RAISE INSUFFICIENT_FUNDS;
    END makePurchase;

    PROCEDURE makeDeposit(p_user IN accounts.username%TYPE, p_sum IN deposits.deposit_sum%TYPE) IS
        v_user_id accounts.account_id%TYPE;
    BEGIN
```

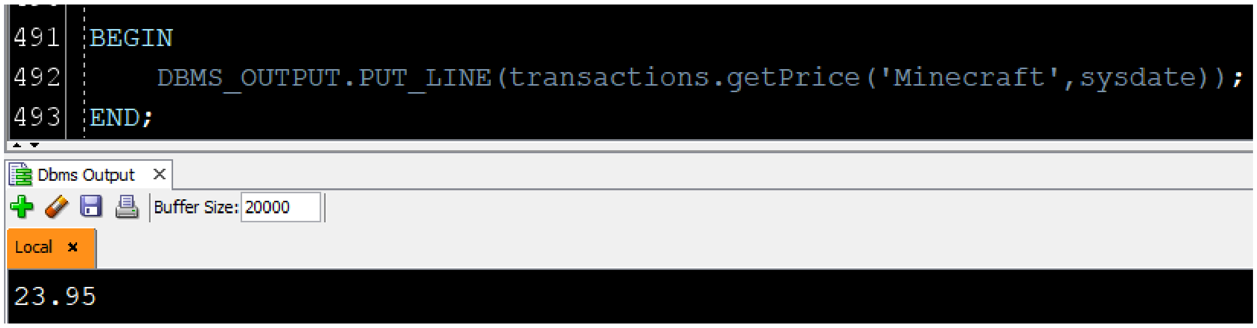
```
        IF p_sum <= 0 THEN RAISE INVALID_SUM;
        END IF;

        SELECT account_id
        INTO v_user_id
        FROM accounts
        WHERE lower(username) = lower(p_user);

        INSERT INTO deposits
        VALUES (v_user_id, sysdate, p_sum);

    EXCEPTION
        WHEN NO_DATA_FOUND THEN DBMS_OUTPUT.PUT_LINE('USER ' || p_user || ' NOT FOUND!');
                                RAISE NO_DATA_FOUND;
        WHEN INVALID_SUM THEN DBMS_OUTPUT.PUT_LINE('DEPOSITED SUM HAS TO BE POSITIVE!');
                              RAISE INVALID_SUM;
    END makeDeposit;
END transactions;
/
```



```
Package TRANSACTIONS compiled


Package Body TRANSACTIONS compiled
```

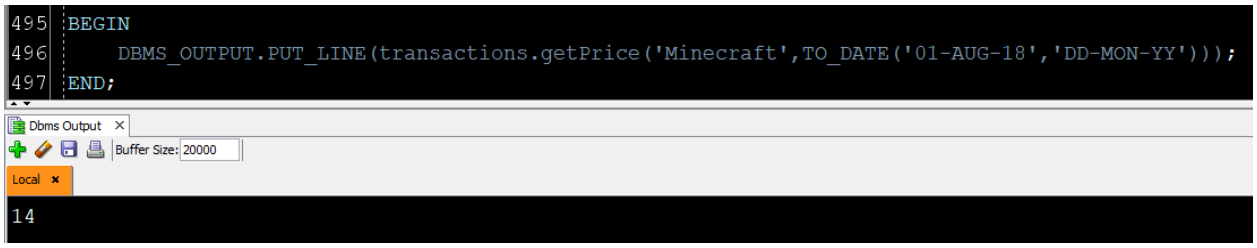
Exemple de utilizare:

Determinarea unui preț(toate cazurile):



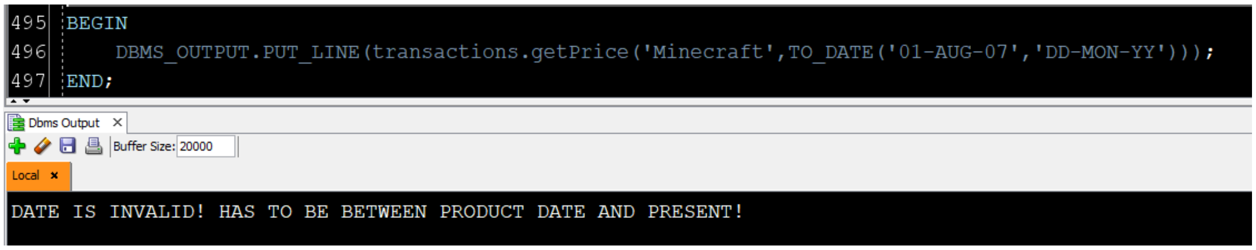
```
BEGIN
    DBMS_OUTPUT.PUT_LINE(transactions.getPrice('Minecraft',sysdate));
END;
```

```
23.95
```



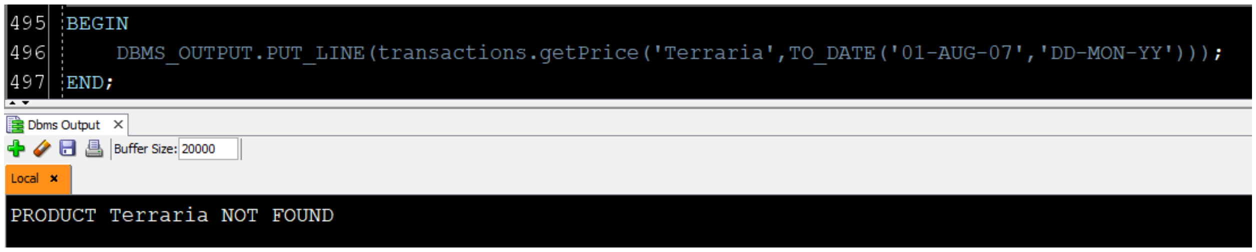
```
BEGIN
    DBMS_OUTPUT.PUT_LINE(transactions.getPrice('Minecraft',TO_DATE('01-AUG-18','DD-MON-YY')));
END;
```

```
14
```



```
BEGIN
    DBMS_OUTPUT.PUT_LINE(transactions.getPrice('Minecraft',TO_DATE('01-AUG-07','DD-MON-YY')));
END;
```

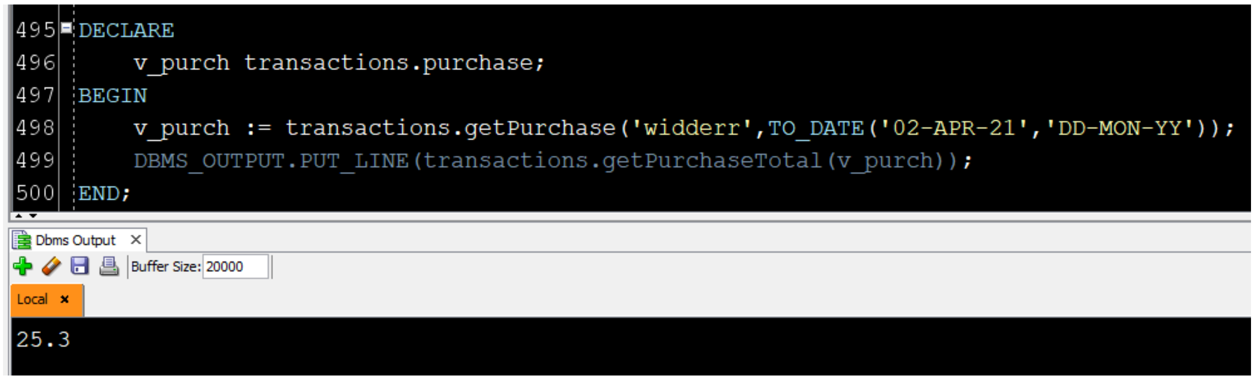
```
DATE IS INVALID! HAS TO BE BETWEEN PRODUCT DATE AND PRESENT!
```



```
BEGIN
    DBMS_OUTPUT.PUT_LINE(transactions.getPrice('Terraria',TO_DATE('01-AUG-07','DD-MON-YY')));
END;
```

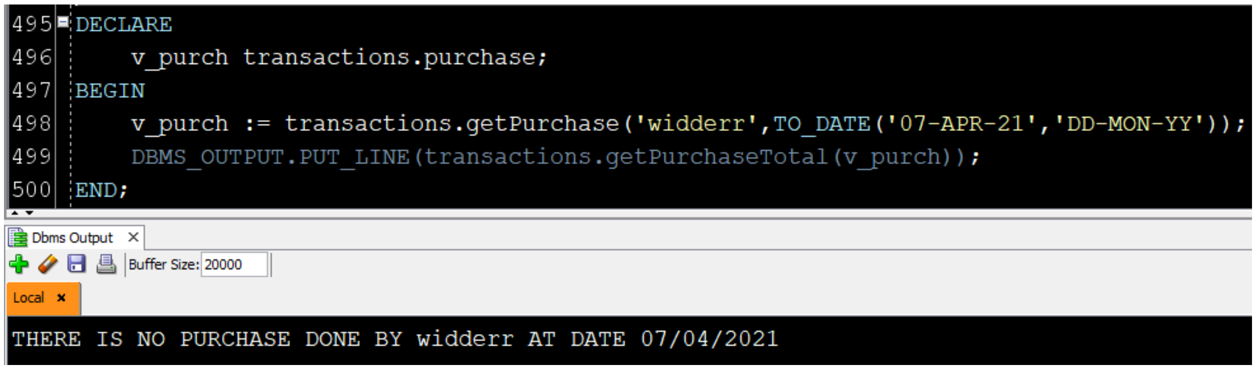
```
PRODUCT Terraria NOT FOUND
```

Găsirea și determinarea valorii unei achiziții:



```
DECLARE
    v_purch transactions.purchase;
BEGIN
    v_purch := transactions.getPurchase('widderr',TO_DATE('02-APR-21','DD-MON-YY'));
    DBMS_OUTPUT.PUT_LINE(transactions.getPurchaseTotal(v_purch));
END;
```

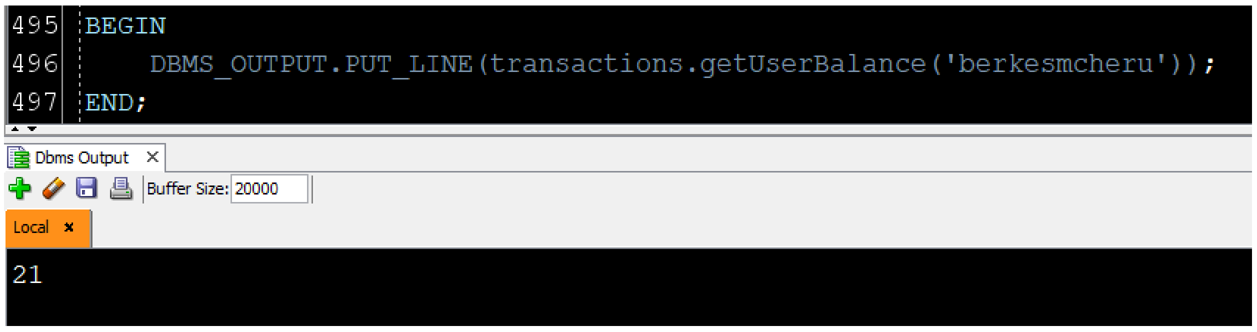
```
25.3
```



```
DECLARE
    v_purch transactions.purchase;
BEGIN
    v_purch := transactions.getPurchase('widderr',TO_DATE('07-APR-21','DD-MON-YY'));
    DBMS_OUTPUT.PUT_LINE(transactions.getPurchaseTotal(v_purch));
END;
```

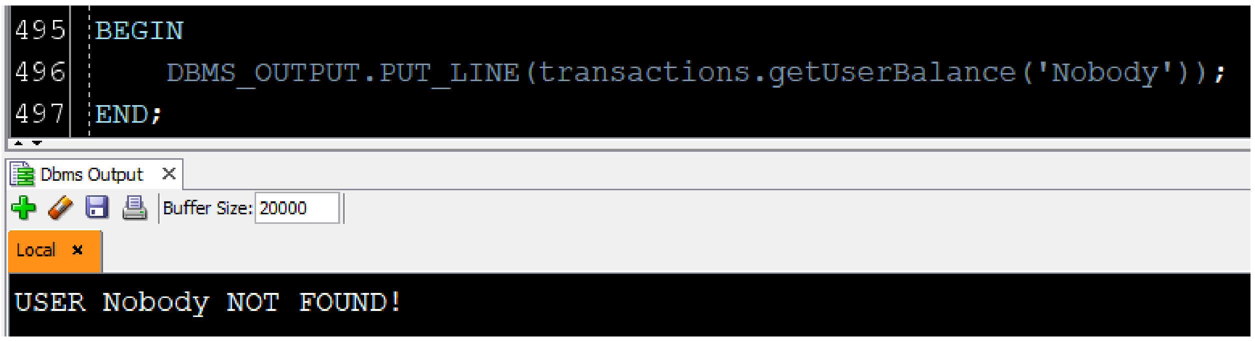
```
THERE IS NO PURCHASE DONE BY widderr AT DATE 07/04/2021
```

Soldul unui utilizator:



```
BEGIN
    DBMS_OUTPUT.PUT_LINE(transactions.getUserBalance('berkesmcheru'));
END;
```

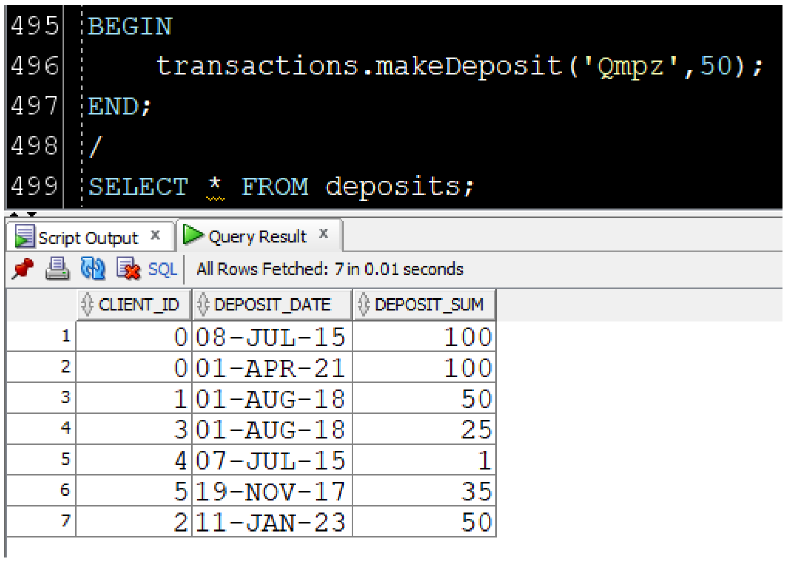
```
21
```



```
BEGIN
    DBMS_OUTPUT.PUT_LINE(transactions.getUserBalance('Nobody'));
END;
```

```
USER Nobody NOT FOUND!
```

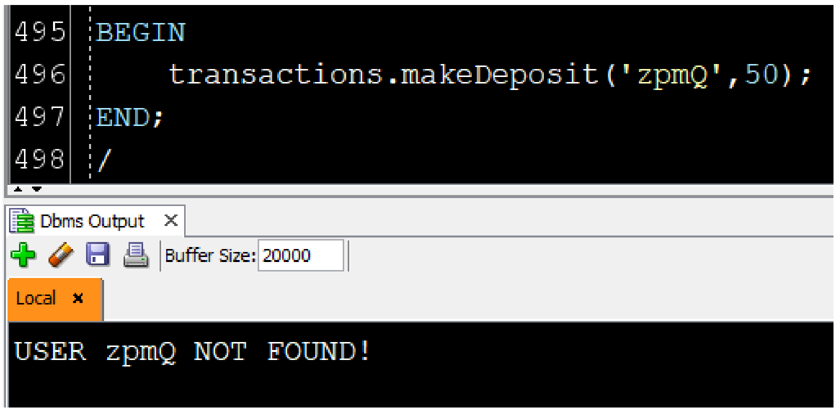
Deposit:



```
BEGIN
    transactions.makeDeposit('Qmpz',50);
END;
/
SELECT * FROM deposits;
```

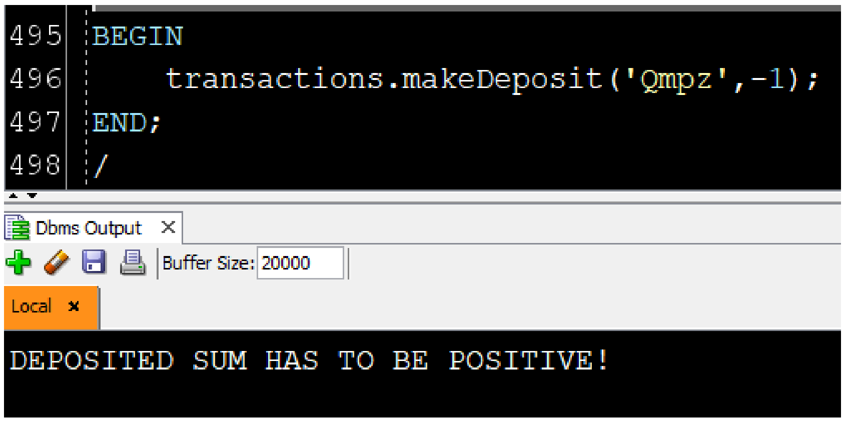
All Rows Fetched: 7 in 0.01 seconds

	CLIENT_ID	DEPOSIT_DATE	DEPOSIT_SUM
1	0	08-JUL-15	100
2	0	01-APR-21	100
3	1	01-AUG-18	50
4	3	01-AUG-18	25
5	4	07-JUL-15	1
6	5	19-NOV-17	35
7	2	11-JAN-23	50



```
BEGIN
    transactions.makeDeposit('zpmQ',50);
END;
/
```

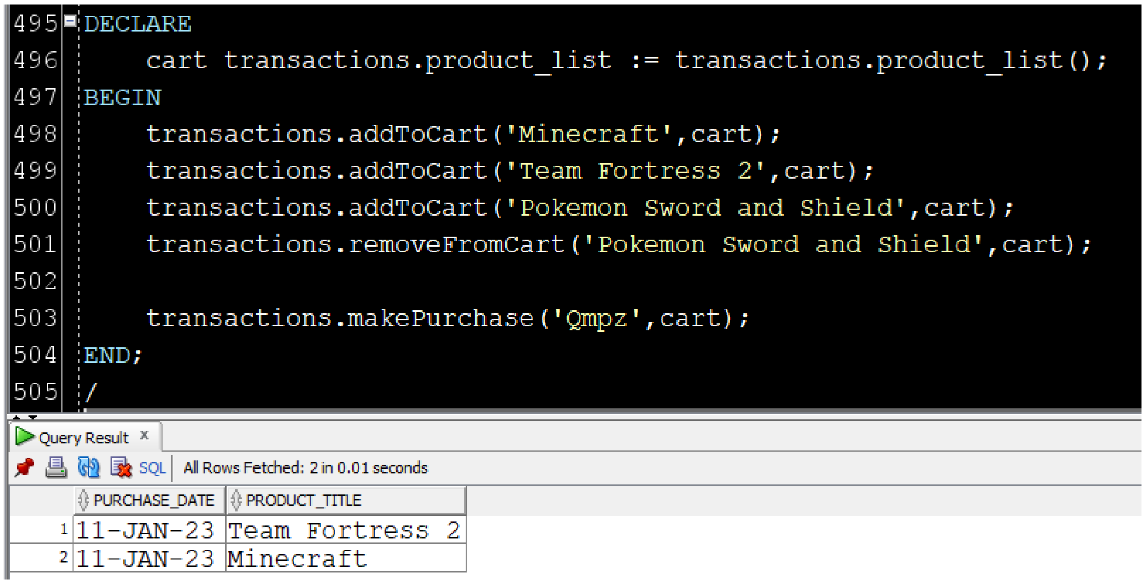
```
USER zpmQ NOT FOUND!
```



```
BEGIN
    transactions.makeDeposit('Qmpz',-1);
END;
/
```

```
DEPOSITED SUM HAS TO BE POSITIVE!
```

Flow de achiziționare:

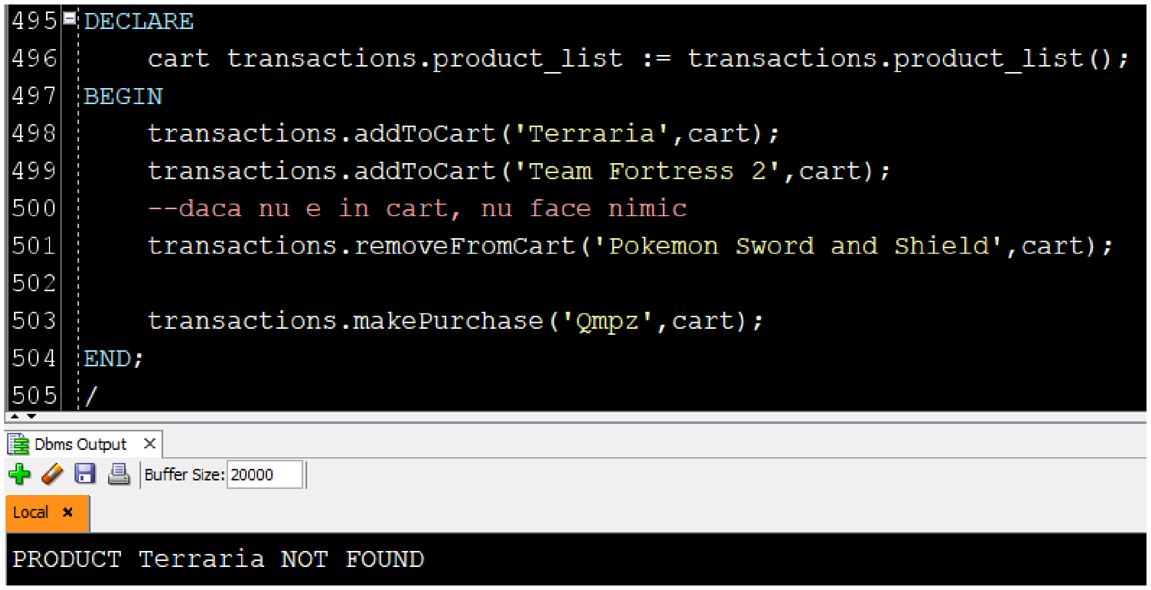


```
DECLARE
    cart transactions.product_list := transactions.product_list();
BEGIN
    transactions.addToCart('Minecraft',cart);
    transactions.addToCart('Team Fortress 2',cart);
    transactions.addToCart('Pokemon Sword and Shield',cart);
    transactions.removeFromCart('Pokemon Sword and Shield',cart);

    transactions.makePurchase('Qmpz',cart);
END;
/
```

Query Result

All Rows Fetched: 2 in 0.01 seconds

	PURCHASE_DATE	PRODUCT_TITLE
1	11-JAN-23	Team Fortress 2
2	11-JAN-23	Minecraft



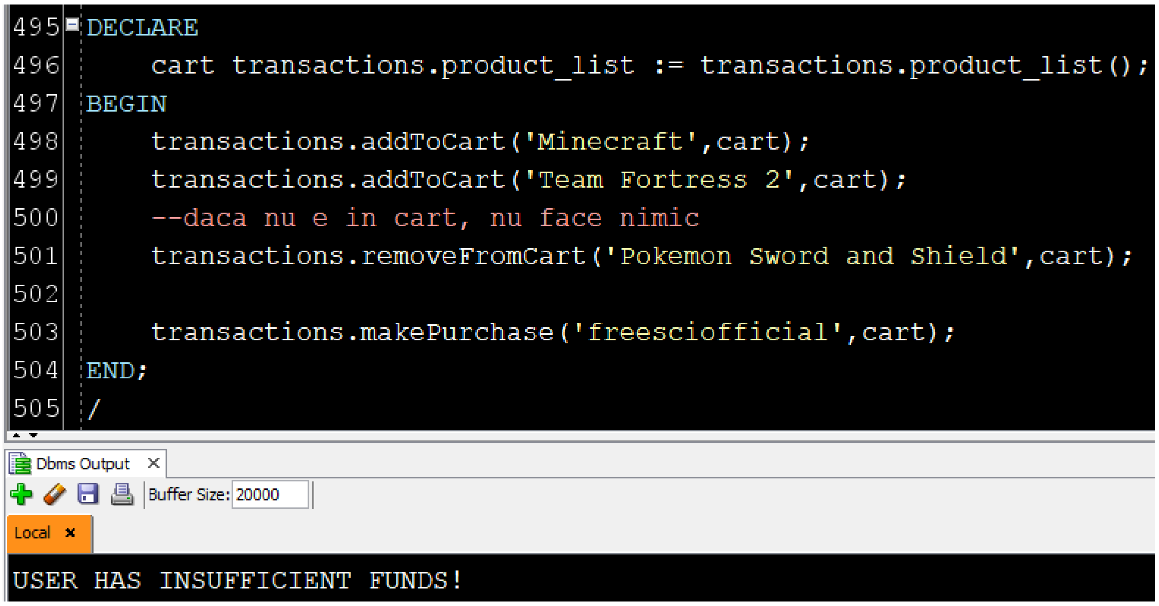
```
DECLARE
    cart transactions.product_list := transactions.product_list();
BEGIN
    transactions.addToCart('Terraria',cart);
    transactions.addToCart('Team Fortress 2',cart);
    --daca nu e in cart, nu face nimic
    transactions.removeFromCart('Pokemon Sword and Shield',cart);

    transactions.makePurchase('Qmpz',cart);
END;
/
```

Dbms Output

Buffer Size: 20000

Local

```
PRODUCT Terraria NOT FOUND
```



```
DECLARE
    cart transactions.product_list := transactions.product_list();
BEGIN
    transactions.addToCart('Minecraft',cart);
    transactions.addToCart('Team Fortress 2',cart);
    --daca nu e in cart, nu face nimic
    transactions.removeFromCart('Pokemon Sword and Shield',cart);

    transactions.makePurchase('freesciofficial',cart);
END;
/
```

Dbms Output

Buffer Size: 20000

Local

```
USER HAS INSUFICIENT FUNDS!
```