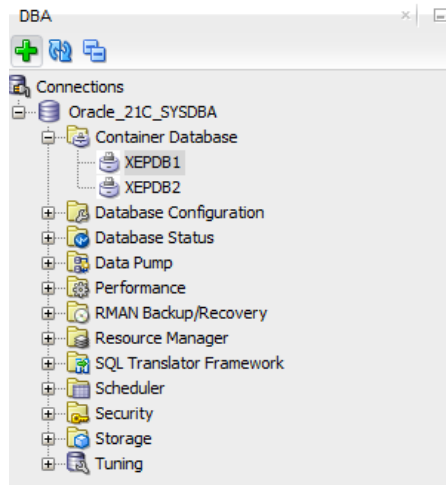


# 1. Crearea bazelor de date și a utilizatorilor

Pentru realizarea proiectului a fost create două baze de date de tip pluggable numite XEPDB1 și XEPDB2.



În baza XEPDB1 au fost creați utilizatori user\_modbd, user\_modbd\_centralizat și user\_modbd\_global, iar în XEPDB2 a fost creat utilizatorul user\_modbd.

-- pentru a creea useri fata c##

alter session set "\_ORACLE\_SCRIPT"=true;

-- user\_modbd

```
CREATE USER user_modbd IDENTIFIED BY Password1;
GRANT CREATE SESSION, CREATE TABLE, CREATE SEQUENCE, CREATE PROCEDURE,
CREATE TRIGGER TO user_modbd;
GRANT ALTER SESSION, ALTER ANY TABLE, ALTER ANY SEQUENCE, ALTER ANY
PROCEDURE, ALTER ANY TRIGGER TO user_modbd;
GRANT SELECT ANY TABLE, INSERT ANY TABLE, UPDATE ANY TABLE, DELETE ANY TABLE TO
user_modbd;
GRANT DROP ANY TABLE, DROP ANY SEQUENCE, DROP ANY PROCEDURE, DROP ANY
TRIGGER TO user_modbd;
GRANT UNLIMITED TABLESPACE TO user_modbd;
GRANT CREATE DATABASE LINK TO user_modbd;
GRANT CREATE PUBLIC DATABASE LINK TO user_modbd;
GRANT DROP PUBLIC DATABASE LINK TO user_modbd;
GRANT RESTRICTED SESSION TO user_modbd;
GRANT ALL PRIVILEGES TO user_modbd;
```

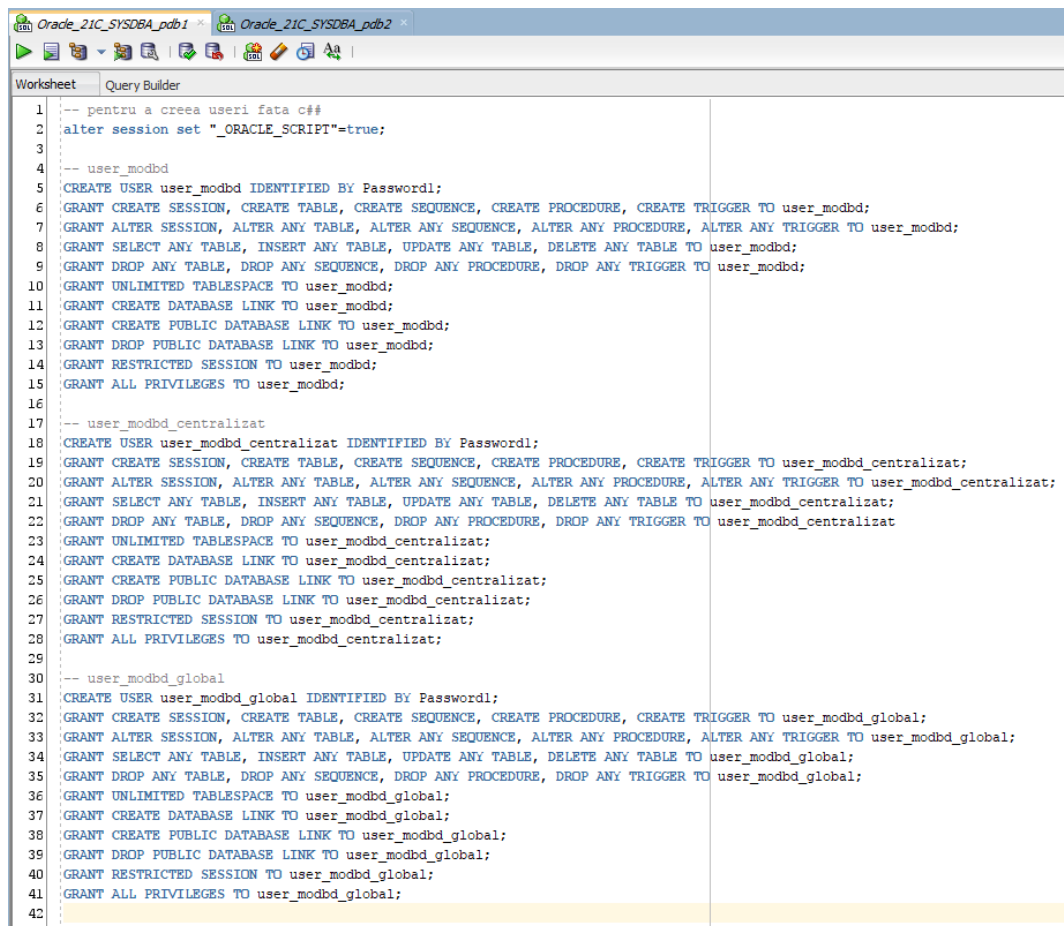
-- user\_modbd\_centralizat

```
CREATE USER user_modbd_centralizat IDENTIFIED BY Password1;
GRANT CREATE SESSION, CREATE TABLE, CREATE SEQUENCE, CREATE PROCEDURE,
CREATE TRIGGER TO user_modbd_centralizat;
GRANT ALTER SESSION, ALTER ANY TABLE, ALTER ANY SEQUENCE, ALTER ANY
PROCEDURE, ALTER ANY TRIGGER TO user_modbd_centralizat;
GRANT SELECT ANY TABLE, INSERT ANY TABLE, UPDATE ANY TABLE, DELETE ANY TABLE TO
user_modbd_centralizat;
GRANT DROP ANY TABLE, DROP ANY SEQUENCE, DROP ANY PROCEDURE, DROP ANY
TRIGGER TO user_modbd_centralizat;
GRANT UNLIMITED TABLESPACE TO user_modbd_centralizat;
GRANT CREATE DATABASE LINK TO user_modbd_centralizat;
GRANT CREATE PUBLIC DATABASE LINK TO user_modbd_centralizat;
GRANT DROP PUBLIC DATABASE LINK TO user_modbd_centralizat;
```

```
GRANT RESTRICTED SESSION TO user_modbd_centralizat;  
GRANT ALL PRIVILEGES TO user_modbd_centralizat;
```

```
-- user_modbd_global
```

```
CREATE USER user_modbd_global IDENTIFIED BY Password1;  
GRANT CREATE SESSION, CREATE TABLE, CREATE SEQUENCE, CREATE PROCEDURE,  
CREATE TRIGGER TO user_modbd_global;  
GRANT ALTER SESSION, ALTER ANY TABLE, ALTER ANY SEQUENCE, ALTER ANY  
PROCEDURE, ALTER ANY TRIGGER TO user_modbd_global;  
GRANT SELECT ANY TABLE, INSERT ANY TABLE, UPDATE ANY TABLE, DELETE ANY TABLE TO  
user_modbd_global;  
GRANT DROP ANY TABLE, DROP ANY SEQUENCE, DROP ANY PROCEDURE, DROP ANY  
TRIGGER TO user_modbd_global;  
GRANT UNLIMITED TABLESPACE TO user_modbd_global;  
GRANT CREATE DATABASE LINK TO user_modbd_global;  
GRANT CREATE PUBLIC DATABASE LINK TO user_modbd_global;  
GRANT DROP PUBLIC DATABASE LINK TO user_modbd_global;  
GRANT RESTRICTED SESSION TO user_modbd_global;  
GRANT ALL PRIVILEGES TO user_modbd_global;
```



```
1  -- pentru a crea useri fara c##  
2  alter session set "_ORACLE_SCRIPT"=true;  
3  
4  -- user_modbd  
5  CREATE USER user_modbd IDENTIFIED BY Password1;  
6  GRANT CREATE SESSION, CREATE TABLE, CREATE SEQUENCE, CREATE PROCEDURE, CREATE TRIGGER TO user_modbd;  
7  GRANT ALTER SESSION, ALTER ANY TABLE, ALTER ANY SEQUENCE, ALTER ANY PROCEDURE, ALTER ANY TRIGGER TO user_modbd;  
8  GRANT SELECT ANY TABLE, INSERT ANY TABLE, UPDATE ANY TABLE, DELETE ANY TABLE TO user_modbd;  
9  GRANT DROP ANY TABLE, DROP ANY SEQUENCE, DROP ANY PROCEDURE, DROP ANY TRIGGER TO user_modbd;  
10 GRANT UNLIMITED TABLESPACE TO user_modbd;  
11 GRANT CREATE DATABASE LINK TO user_modbd;  
12 GRANT CREATE PUBLIC DATABASE LINK TO user_modbd;  
13 GRANT DROP PUBLIC DATABASE LINK TO user_modbd;  
14 GRANT RESTRICTED SESSION TO user_modbd;  
15 GRANT ALL PRIVILEGES TO user_modbd;  
16  
17 -- user_modbd_centralizat  
18 CREATE USER user_modbd_centralizat IDENTIFIED BY Password1;  
19 GRANT CREATE SESSION, CREATE TABLE, CREATE SEQUENCE, CREATE PROCEDURE, CREATE TRIGGER TO user_modbd_centralizat;  
20 GRANT ALTER SESSION, ALTER ANY TABLE, ALTER ANY SEQUENCE, ALTER ANY PROCEDURE, ALTER ANY TRIGGER TO user_modbd_centralizat;  
21 GRANT SELECT ANY TABLE, INSERT ANY TABLE, UPDATE ANY TABLE, DELETE ANY TABLE TO user_modbd_centralizat;  
22 GRANT DROP ANY TABLE, DROP ANY SEQUENCE, DROP ANY PROCEDURE, DROP ANY TRIGGER TO user_modbd_centralizat;  
23 GRANT UNLIMITED TABLESPACE TO user_modbd_centralizat;  
24 GRANT CREATE DATABASE LINK TO user_modbd_centralizat;  
25 GRANT CREATE PUBLIC DATABASE LINK TO user_modbd_centralizat;  
26 GRANT DROP PUBLIC DATABASE LINK TO user_modbd_centralizat;  
27 GRANT RESTRICTED SESSION TO user_modbd_centralizat;  
28 GRANT ALL PRIVILEGES TO user_modbd_centralizat;  
29  
30 -- user_modbd_global  
31 CREATE USER user_modbd_global IDENTIFIED BY Password1;  
32 GRANT CREATE SESSION, CREATE TABLE, CREATE SEQUENCE, CREATE PROCEDURE, CREATE TRIGGER TO user_modbd_global;  
33 GRANT ALTER SESSION, ALTER ANY TABLE, ALTER ANY SEQUENCE, ALTER ANY PROCEDURE, ALTER ANY TRIGGER TO user_modbd_global;  
34 GRANT SELECT ANY TABLE, INSERT ANY TABLE, UPDATE ANY TABLE, DELETE ANY TABLE TO user_modbd_global;  
35 GRANT DROP ANY TABLE, DROP ANY SEQUENCE, DROP ANY PROCEDURE, DROP ANY TRIGGER TO user_modbd_global;  
36 GRANT UNLIMITED TABLESPACE TO user_modbd_global;  
37 GRANT CREATE DATABASE LINK TO user_modbd_global;  
38 GRANT CREATE PUBLIC DATABASE LINK TO user_modbd_global;  
39 GRANT DROP PUBLIC DATABASE LINK TO user_modbd_global;  
40 GRANT RESTRICTED SESSION TO user_modbd_global;  
41 GRANT ALL PRIVILEGES TO user_modbd_global;  
42
```

```

1  -- pentru a crea useri fata c##
2  alter session set "_ORACLE_SCRIPT"=true;
3
4  -- user_modbd
5  CREATE USER user_modbd IDENTIFIED BY Password1;
6  GRANT CREATE SESSION, CREATE TABLE, CREATE SEQUENCE, CREATE PROCEDURE, CREATE TRIGGER TO user_modbd;
7  GRANT ALTER SESSION, ALTER ANY TABLE, ALTER ANY SEQUENCE, ALTER ANY PROCEDURE, ALTER ANY TRIGGER TO user_modbd;
8  GRANT SELECT ANY TABLE, INSERT ANY TABLE, UPDATE ANY TABLE, DELETE ANY TABLE TO user_modbd;
9  GRANT DROP ANY TABLE, DROP ANY SEQUENCE, DROP ANY PROCEDURE, DROP ANY TRIGGER TO user_modbd;
10 GRANT UNLIMITED TABLESPACE TO user_modbd;
11 GRANT CREATE DATABASE LINK TO user_modbd;
12 GRANT CREATE PUBLIC DATABASE LINK TO user_modbd;
13 GRANT DROP PUBLIC DATABASE LINK TO user_modbd;
14 GRANT RESTRICTED SESSION TO user_modbd;
15 GRANT ALL PRIVILEGES TO user_modbd;
16

```

Apoi am ne-am conectat ca user\_modbd la pdb1 și pdb2 și am creat link-urile dintre acestea

--creare link pdb2

```

CREATE PUBLIC DATABASE LINK pdb2
CONNECT TO user_modbd
IDENTIFIED BY Password1
USING '(DESCRIPTION=
        (ADDRESS=(PROTOCOL=TCP)(HOST=localhost)(PORT=1521))
        (CONNECT_DATA=(SERVICE_NAME=xepdb2))
)';

```

--creare link pdb1

```

CREATE PUBLIC DATABASE LINK pdb1
CONNECT TO user_modbd
IDENTIFIED BY Password1
USING '(DESCRIPTION=
        (ADDRESS=(PROTOCOL=TCP)(HOST=localhost)(PORT=1521))
        (CONNECT_DATA=(SERVICE_NAME=xepdb1))
)';

```

```

1  --create link pdb2
2  CREATE PUBLIC DATABASE LINK pdb2
3    CONNECT TO user_modbd
4    IDENTIFIED BY Password1
5  USING '(DESCRIPTION=
6        (ADDRESS=(PROTOCOL=TCP)(HOST=localhost)(PORT=1521))
7        (CONNECT_DATA=(SERVICE_NAME=xepdb2))
8    )';

```

Script Output x

Task completed in 0.091 seconds

Database link PDB2 created.

```

1  --create link pdb1
2  CREATE PUBLIC DATABASE LINK pdb1
3    CONNECT TO user_modbd
4    IDENTIFIED BY Password1
5  USING '(DESCRIPTION=
6        (ADDRESS=(PROTOCOL=TCP)(HOST=localhost)(PORT=1521))
7        (CONNECT_DATA=(SERVICE_NAME=xepdb1))
8    )';

```

Script Output x

Task completed in 0.051 seconds

Database link PDB1 created.

Următorul pas a fost crearea tabelelor în baza de date centralizat.

--creare bd centralizat

```

CREATE TABLE cities_all (
    id int,
    name varchar(255),

```

```

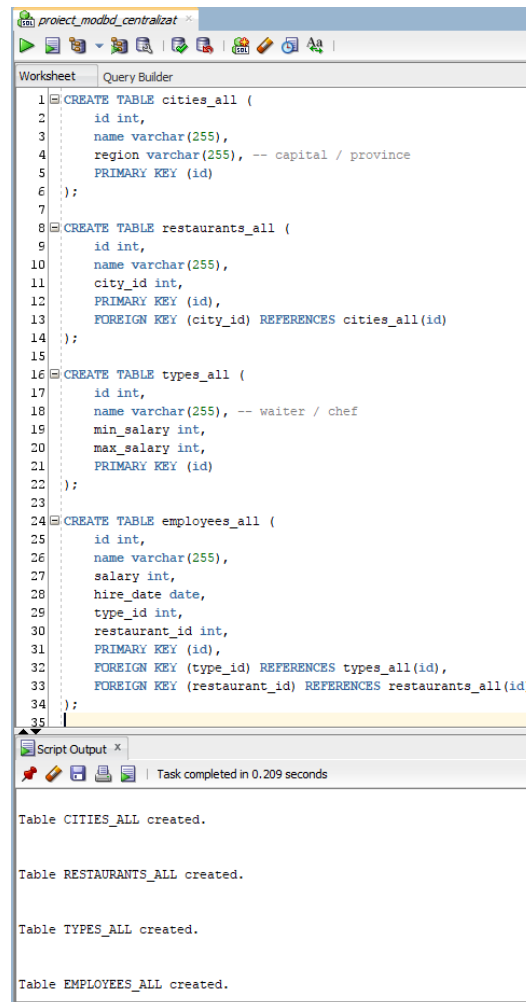
    region varchar(255), -- capital / province
    PRIMARY KEY (id)
);
CREATE TABLE restaurants_all (
    id int,
    name varchar(255),
    city_id int,
    PRIMARY KEY (id),
    FOREIGN KEY (city_id) REFERENCES cities_all(id)
);
CREATE TABLE types_all (
    id int,
    name varchar(255), -- waiter / chef
    min_salary int,
    max_salary int,
    PRIMARY KEY (id)
);
CREATE TABLE employees_all (
    id int,
    name varchar(255),
    salary int,
    hire_date date,
    type_id int,
    restaurant_id int,
    PRIMARY KEY (id),
    FOREIGN KEY (type_id) REFERENCES types_all(id),
    FOREIGN KEY (restaurant_id) REFERENCES restaurants_all(id)
);
CREATE TABLE menus_all (
    id int,
    menu_date date,
    PRIMARY KEY (id)
);
CREATE TABLE dishes_all (
    id int,
    name varchar(255),
    price int,
    menu_id int,
    PRIMARY KEY (id),
    FOREIGN KEY (menu_id) REFERENCES menus_all(id)
);
CREATE TABLE drinks_all (
    id int,
    name varchar(255),
    type varchar(255), -- alcoholic / non-alcoholic
    price int,
    menu_id int,
    PRIMARY KEY (id),
    FOREIGN KEY (menu_id) REFERENCES menus_all(id)
);
CREATE TABLE orders_all (
    id int,
    order_date date,
    total int,
    tip int,
    waiter_id int,
    PRIMARY KEY (id),
    FOREIGN KEY (waiter_id) REFERENCES employees_all(id)
);
CREATE TABLE orders_drinks_all (

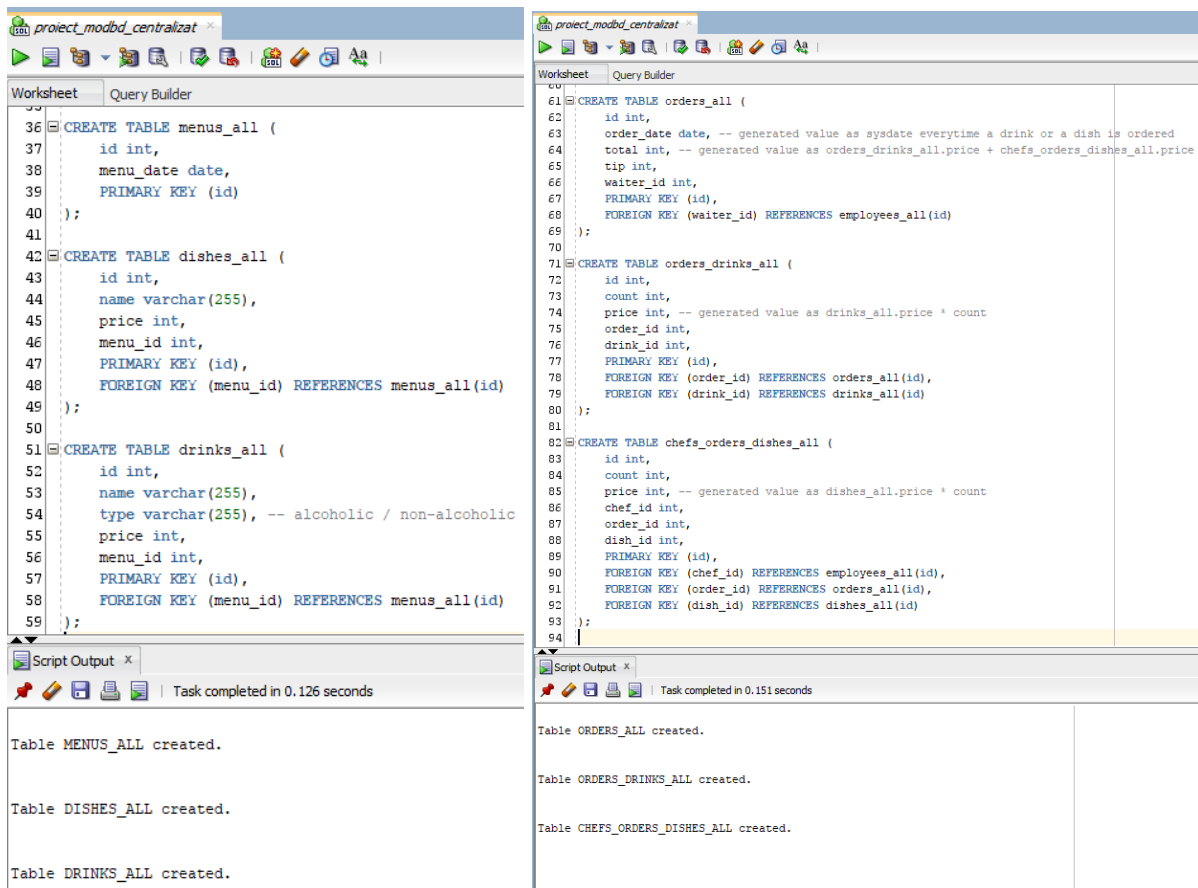
```

```

id int,
count int,
price int,
order_id int,
drink_id int,
PRIMARY KEY (id),
FOREIGN KEY (order_id) REFERENCES orders_all(id),
FOREIGN KEY (drink_id) REFERENCES drinks_all(id)
);
CREATE TABLE chefs_orders_dishes_all (
id int,
count int,
price int,
chef_id int,
order_id int,
dish_id int,
PRIMARY KEY (id),
FOREIGN KEY (chef_id) REFERENCES employees_all(id),
FOREIGN KEY (order_id) REFERENCES orders_all(id),
FOREIGN KEY (dish_id) REFERENCES dishes_all(id)
);

```





Pentru valorile generate am folosit următoarele triggere:

```
-- valori_generate
CREATE OR REPLACE TRIGGER trigger_orders_drinks_all_price
  BEFORE INSERT OR UPDATE OR DELETE ON orders_drinks_all
  FOR EACH ROW
DECLARE
  v_price INT;
BEGIN
  IF INSERTING THEN
    SELECT price INTO v_price
    FROM drinks_all
    WHERE id = :NEW.drink_id;
    :NEW.price := :NEW.count * v_price;
    UPDATE orders_all
    SET total = total + :NEW.price
    WHERE id = :NEW.order_id;
  ELSIF UPDATING THEN
    SELECT price INTO v_price
    FROM drinks_all
    WHERE id = :NEW.drink_id;
    :NEW.price := :NEW.count * v_price;
    UPDATE orders_all
    SET total = total - :OLD.price + :NEW.price
    WHERE id = :NEW.order_id;
  ELSIF DELETING THEN
    SELECT price INTO v_price
    FROM drinks_all
    WHERE id = :OLD.drink_id;
    UPDATE orders_all
    SET total = total - v_price * :OLD.count
```

```

        WHERE id = :OLD.order_id;
    END IF;
END;
/
CREATE OR REPLACE TRIGGER trigger_chefs_orders_dishes_all_price
    BEFORE INSERT OR UPDATE OR DELETE ON chefs_orders_dishes_all
    FOR EACH ROW
DECLARE
    v_price INT;
BEGIN
    IF INSERTING THEN
        SELECT price INTO v_price
        FROM dishes_all
        WHERE id = :NEW.dish_id;
        :NEW.price := :NEW.count * v_price;
        UPDATE orders_all
        SET total = total + :NEW.price
        WHERE id = :NEW.order_id;
    ELSIF UPDATING THEN
        SELECT price INTO v_price
        FROM dishes_all
        WHERE id = :NEW.dish_id;
        :NEW.price := :NEW.count * v_price;
        UPDATE orders_all
        SET total = total - :OLD.price + :NEW.price
        WHERE id = :NEW.order_id;
    ELSIF DELETING THEN
        SELECT price INTO v_price
        FROM dishes_all
        WHERE id = :OLD.dish_id;
        UPDATE orders_all
        SET total = total - v_price * :OLD.count
        WHERE id = :OLD.order_id;
    END IF;
END;
/
CREATE OR REPLACE TRIGGER trigger_orders_all_date
    BEFORE INSERT OR UPDATE ON orders_all
    FOR EACH ROW
DECLARE
    v_date DATE;
BEGIN
    IF INSERTING THEN
        SELECT sysdate INTO v_date
        FROM dual;
        :NEW.order_date := v_date;
        :NEW.total := 0;
    ELSIF UPDATING THEN
        SELECT sysdate INTO v_date
        FROM dual;
        :NEW.order_date := v_date;
    END IF;
END;
/

```

proiect\_modbd\_centralizat

Worksheet Query Builder

```

1 -- valori_generate
2 CREATE OR REPLACE TRIGGER trigger_orders_drinks_all_price
3 BEFORE INSERT OR UPDATE OR DELETE ON orders_drinks_all
4 FOR EACH ROW
5 DECLARE
6     v_price INT;
7 BEGIN
8     IF INSERTING THEN
9         SELECT price INTO v_price
10        FROM drinks_all
11        WHERE id = :NEW.drink_id;
12        :NEW.price := :NEW.count * v_price;
13        UPDATE orders_all
14        SET total = total + :NEW.price
15        WHERE id = :NEW.order_id;
16    ELSIF UPDATING THEN
17        SELECT price INTO v_price
18        FROM drinks_all
19        WHERE id = :NEW.drink_id;
20        :NEW.price := :NEW.count * v_price;
21        UPDATE orders_all
22        SET total = total - :OLD.price + :NEW.price
23        WHERE id = :NEW.order_id;
24    ELSIF DELETING THEN
25        SELECT price INTO v_price
26        FROM drinks_all
27        WHERE id = :OLD.drink_id;
28        UPDATE orders_all
29        SET total = total - v_price * :OLD.count
30        WHERE id = :OLD.order_id;
31    END IF;
32 END;
33 /

```

Script Output x

Task completed in 0.062 seconds

Trigger TRIGGER\_ORDERS\_DRINKS\_ALL\_PRICE compiled

proiect\_modbd\_centralizat

Worksheet Query Builder

```

34
35 CREATE OR REPLACE TRIGGER trigger_chefs_orders_dishes_all_price
36 BEFORE INSERT OR UPDATE OR DELETE ON chefs_orders_dishes_all
37 FOR EACH ROW
38 DECLARE
39     v_price INT;
40 BEGIN
41     IF INSERTING THEN
42         SELECT price INTO v_price
43         FROM dishes_all
44         WHERE id = :NEW.dish_id;
45         :NEW.price := :NEW.count * v_price;
46         UPDATE orders_all
47         SET total = total + :NEW.price
48         WHERE id = :NEW.order_id;
49     ELSIF UPDATING THEN
50         SELECT price INTO v_price
51         FROM dishes_all
52         WHERE id = :NEW.dish_id;
53         :NEW.price := :NEW.count * v_price;
54         UPDATE orders_all
55         SET total = total - :OLD.price + :NEW.price
56         WHERE id = :NEW.order_id;
57     ELSIF DELETING THEN
58         SELECT price INTO v_price
59         FROM dishes_all
60         WHERE id = :OLD.dish_id;
61         UPDATE orders_all
62         SET total = total - v_price * :OLD.count
63         WHERE id = :OLD.order_id;
64     END IF;
65 END;
66 /
67

```

Script Output x

Task completed in 0.081 seconds

Trigger TRIGGER\_ORDERS\_DRINKS\_ALL\_PRICE compiled

Trigger TRIGGER\_CHEFS\_ORDERS\_DISHES\_ALL\_PRICE compiled

proiect\_modbd\_centralizat

Worksheet Query Builder

```

67
68 CREATE OR REPLACE TRIGGER trigger_orders_all_date
69 BEFORE INSERT OR UPDATE ON orders_all
70 FOR EACH ROW
71 DECLARE
72     v_date DATE;
73 BEGIN
74     IF INSERTING THEN
75         SELECT sysdate INTO v_date
76         FROM dual;
77         :NEW.order_date := v_date;
78         :NEW.total := 0;
79     ELSIF UPDATING THEN
80         SELECT sysdate INTO v_date
81         FROM dual;
82         :NEW.order_date := v_date;
83     END IF;
84 END;
85 /

```

Script Output x

Task completed in 0.07 seconds

Trigger TRIGGER\_ORDERS\_DRINKS\_ALL\_PRICE compiled

Trigger TRIGGER\_CHEFS\_ORDERS\_DISHES\_ALL\_PRICE compiled

Trigger TRIGGER\_ORDERS\_ALL\_DATE compiled



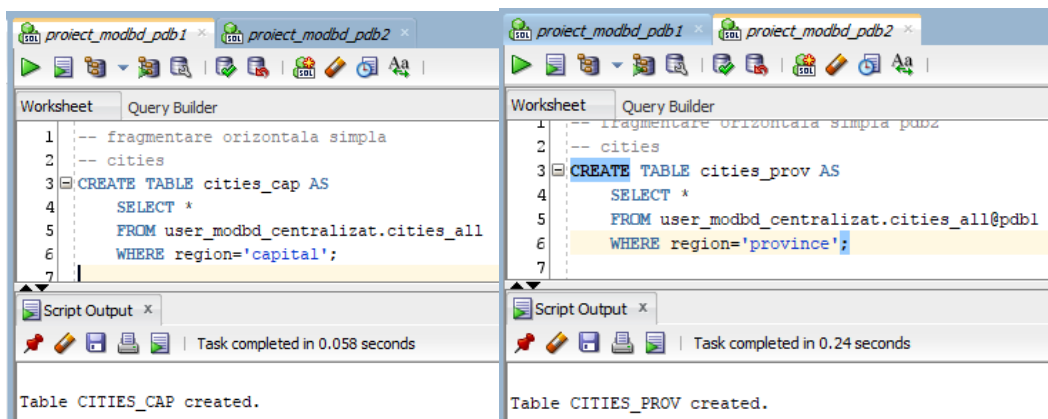
## 2. Crearea relațiilor și a fragmentelor

### a. Fragmentare orizontală primară

Fragmentarea orizontală primară am ales tabela cities\_all, în funcție de atributul region. Astfel, în PDB1 vom avea cities\_cap, unde vor fi stocate orașele capitale, iar în PDB2 în cities\_prov vor fi stocate orașele din provincie.

```
CREATE TABLE cities_cap AS
SELECT *
FROM user_modbd_centralizat.cities_all
WHERE region='capital';
```

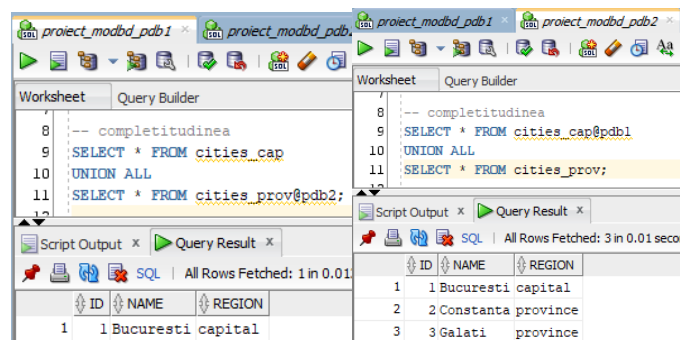
```
CREATE TABLE cities_prov AS
SELECT *
FROM user_modbd_centralizat.cities_all@pdb1
WHERE region='province';
```



Pentru testarea corectitudinii fragmentării, am realizat operațiile de completitudine, reconstrucție și disjuncție

```
-- completitudinea
SELECT * FROM cities_cap
UNION ALL
SELECT * FROM cities_prov@pdb2;
```

```
SELECT * FROM cities_cap@pdb1
UNION ALL
SELECT * FROM cities_prov;
```



```

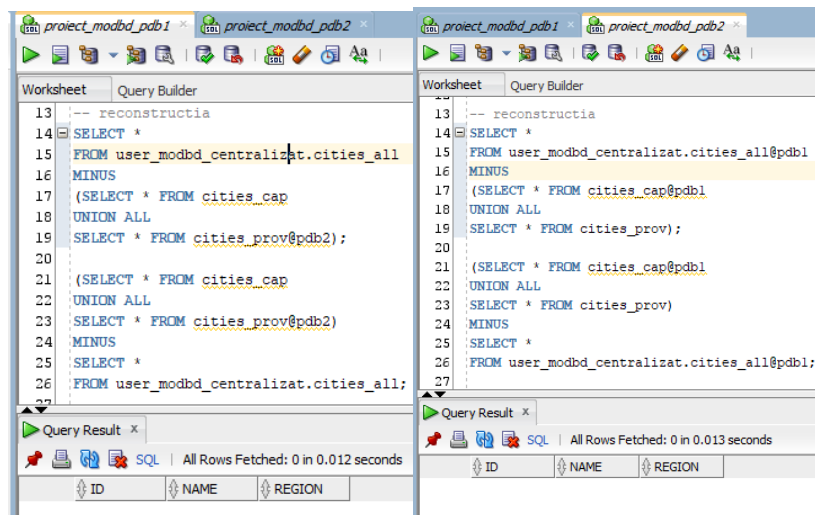
-- restructia
SELECT *
FROM user_modbd_centralizat.cities_all
MINUS
(SELECT * FROM cities_cap
UNION ALL
SELECT * FROM cities_prov@pdb2);

(SELECT * FROM cities_cap
UNION ALL
SELECT * FROM cities_prov@pdb2)
MINUS
SELECT *
FROM user_modbd_centralizat.cities_all;

SELECT *
FROM user_modbd_centralizat.cities_all@pdb1
MINUS
(SELECT * FROM cities_cap@pdb1
UNION ALL
SELECT * FROM cities_prov);

(SELECT * FROM cities_cap@pdb1
UNION ALL
SELECT * FROM cities_prov)
MINUS
SELECT *
FROM user_modbd_centralizat.cities_all@pdb1;

```

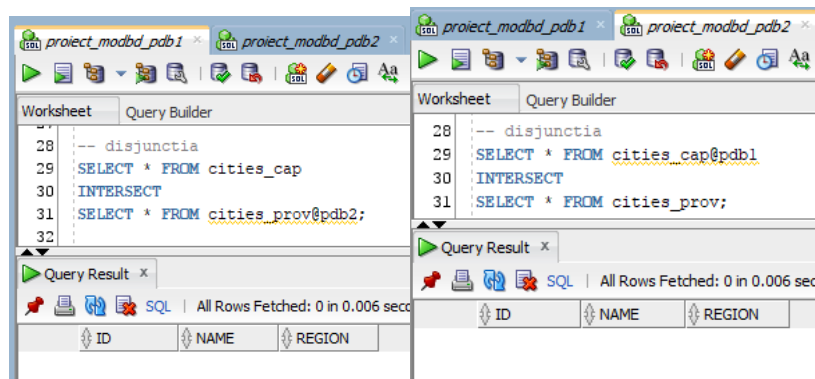


```

-- disjunctia
SELECT * FROM cities_cap
INTERSECT
SELECT * FROM cities_prov@pdb2;

SELECT * FROM cities_cap@pdb1
INTERSECT
SELECT * FROM cities_prov;

```



## b. Fragmentare orizontală derivată

Fragmentarea orizontală derivată a fost realizată asupra tabelelor restaurants, employees, orders, orders\_drinks și chef\_orders\_dishes.

-- fragmentare orizontala derivata pdb1

-- restaurants

CREATE TABLE restaurants\_cap

AS

SELECT r.\*

FROM user\_modbd\_centralizat.restaurants\_all r

WHERE EXISTS

(SELECT \*

FROM cities\_cap c

WHERE r.city\_id = c.id

);

-- employees

CREATE TABLE employees\_cap

AS

SELECT e.\*

FROM user\_modbd\_centralizat.employees\_all e

WHERE EXISTS

(SELECT \*

FROM restaurants\_cap r

WHERE e.restaurant\_id = r.id

);

-- orders

CREATE TABLE orders\_cap

AS

SELECT o.\*

FROM user\_modbd\_centralizat.orders\_all o

WHERE EXISTS

(SELECT \*

FROM employees\_cap e

WHERE o.waiter\_id = e.id

);

-- orders\_drinks

CREATE TABLE orders\_drinks\_cap

AS

SELECT od.\*

FROM user\_modbd\_centralizat.orders\_drinks\_all od

WHERE EXISTS

```

(SELECT *
FROM orders_cap o
WHERE od.order_id = o.id
);

-- chefs_orders_dishes
CREATE TABLE chefs_orders_dishes_cap
AS
SELECT cod.*
FROM user_modbd_centralizat.chefs_orders_dishes_all cod
WHERE EXISTS
(SELECT *
FROM orders_cap o, employees_cap e
WHERE cod.order_id = o.id AND cod.chef_id = e.id
);

```

The image displays two side-by-side screenshots of the SQL Developer interface, showing the execution of SQL scripts to create tables in a database.

**Left Window (project\_modbd\_pdb1):**

- Query Builder:** Shows the SQL script for creating tables. The script includes comments for "fragmentare orizontala derivata" and "restaurants". It defines `restaurants_cap` as a table containing data from `restaurants_all` joined with `cities_cap`. It also defines `employees_cap` as a table containing data from `employees_all` joined with `restaurants_cap`. Finally, it defines `orders_cap` as a table containing data from `orders_all` joined with `employees_cap`.
- Script Output:** Shows the results of the script execution. The output indicates that the tables `RESTAURANTS_CAP`, `EMPLOYEES_CAP`, `ORDERS_CAP`, and `ORDERS_DRINKS_CAP` were successfully created.

**Right Window (project\_modbd\_pdb2):**

- Query Builder:** Shows the SQL script for creating the `chefs_orders_dishes_cap` table. The script defines this table as containing data from `chefs_orders_dishes_all` joined with `orders_cap` and `employees_cap` based on specific order and chef IDs.
- Script Output:** Shows the results of the script execution. The output indicates that the table `CHEFS_ORDERS_DISHES_CAP` was successfully created.

```

-- fragmentare orizontala derivata pdb2
-- restaurants
CREATE TABLE restaurants_prov
AS
SELECT r.*
FROM user_modbd_centralizat.restaurants_all@pdb1 r
WHERE EXISTS
  (SELECT *
   FROM cities_prov c
   WHERE r.city_id = c.id
  );

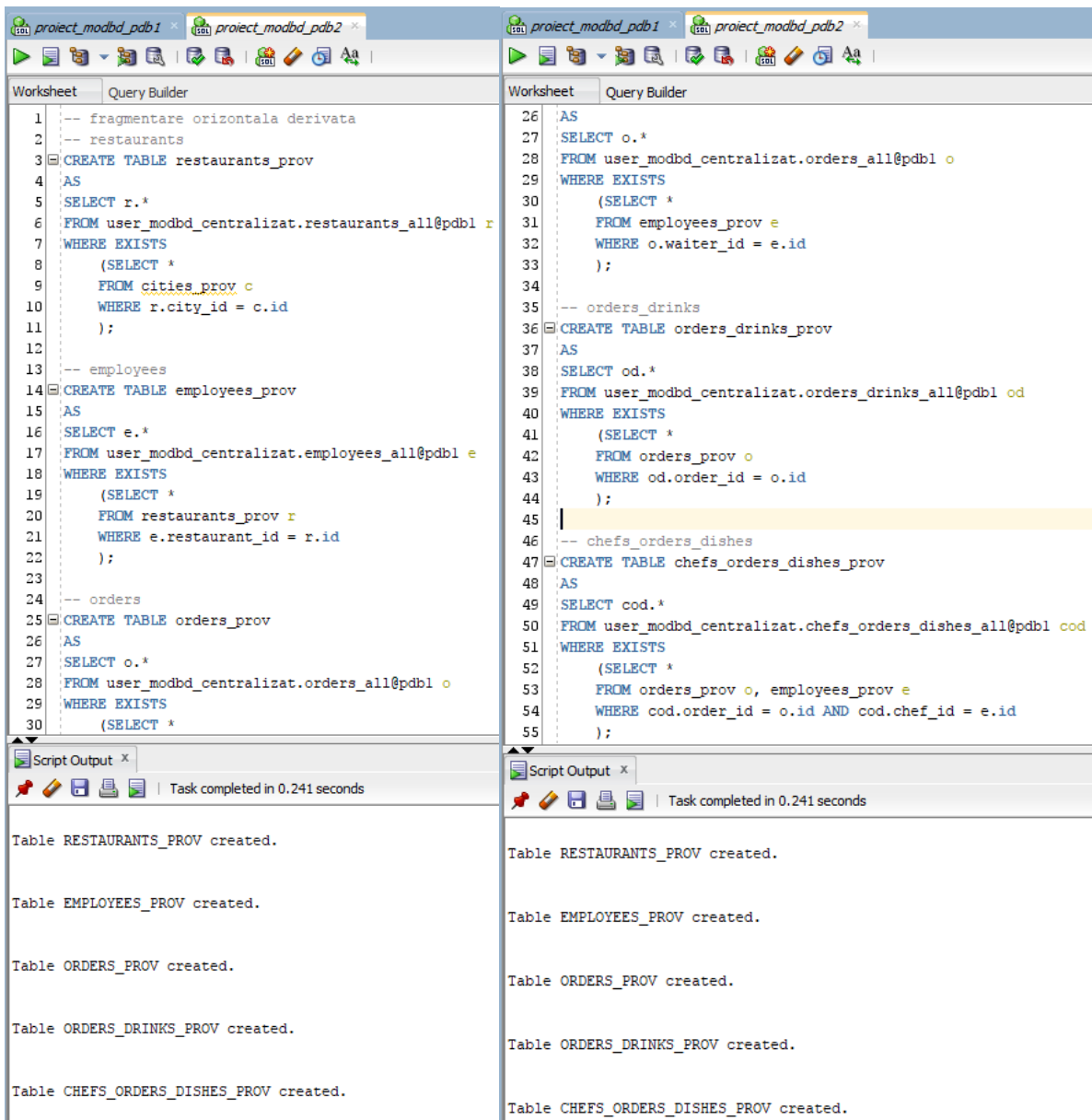
-- employees
CREATE TABLE employees_prov
AS
SELECT e.*
FROM user_modbd_centralizat.employees_all@pdb1 e
WHERE EXISTS
  (SELECT *
   FROM restaurants_prov r
   WHERE e.restaurant_id = r.id
  );

-- orders
CREATE TABLE orders_prov
AS
SELECT o.*
FROM user_modbd_centralizat.orders_all@pdb1 o
WHERE EXISTS
  (SELECT *
   FROM employees_prov e
   WHERE o.waiter_id = e.id
  );

-- orders_drinks
CREATE TABLE orders_drinks_prov
AS
SELECT od.*
FROM user_modbd_centralizat.orders_drinks_all@pdb1 od
WHERE EXISTS
  (SELECT *
   FROM orders_prov o
   WHERE od.order_id = o.id
  );

-- chefs_orders_dishes
CREATE TABLE chefs_orders_dishes_prov
AS
SELECT cod.*
FROM user_modbd_centralizat.chefs_orders_dishes_all@pdb1 cod
WHERE EXISTS
  (SELECT *
   FROM orders_prov o, employees_prov e
   WHERE cod.order_id = o.id AND cod.chef_id = e.id
  );

```



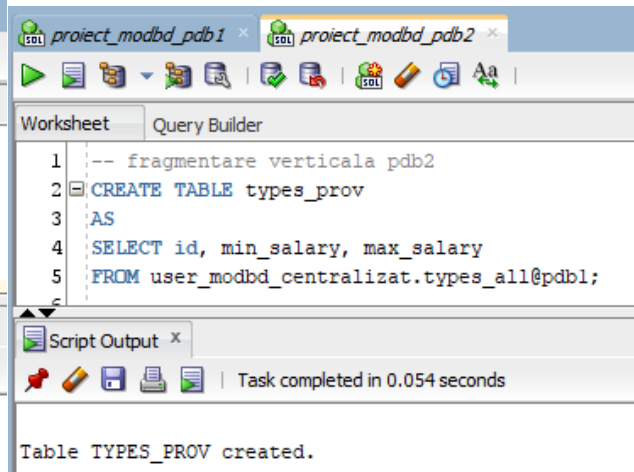
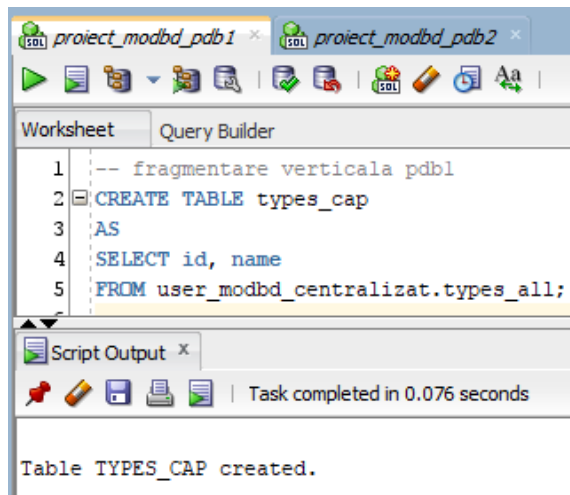
### c. Fragmentare verticală

Pentru fragmentarea verticală, am ales tabela `types`, motivul fiind doar unul demonstrativ. Astfel, în tabela `types_cap` vom avea coloanele `id` și `name`, iar în `types_prov` vor fi `id`, `min_salary` și `max_salary`

```

-- fragmentare verticala pdb1
CREATE TABLE types_cap
AS
SELECT id, name
FROM user_modbd_centralizat.types_all;

```



```

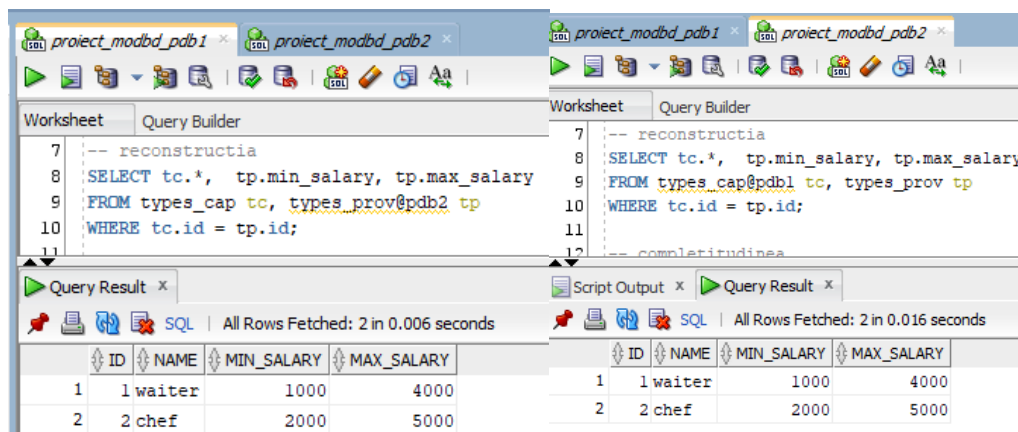
-- reconstructia
SELECT tc.*, tp.min_salary, tp.max_salary
FROM types_cap tc, types_prov@pdb2 tp
WHERE tc.id = tp.id;

```

```

SELECT tc.*, tp.min_salary, tp.max_salary
FROM types_cap@pdb1 tc, types_prov tp
WHERE tc.id = tp.id;

```



```

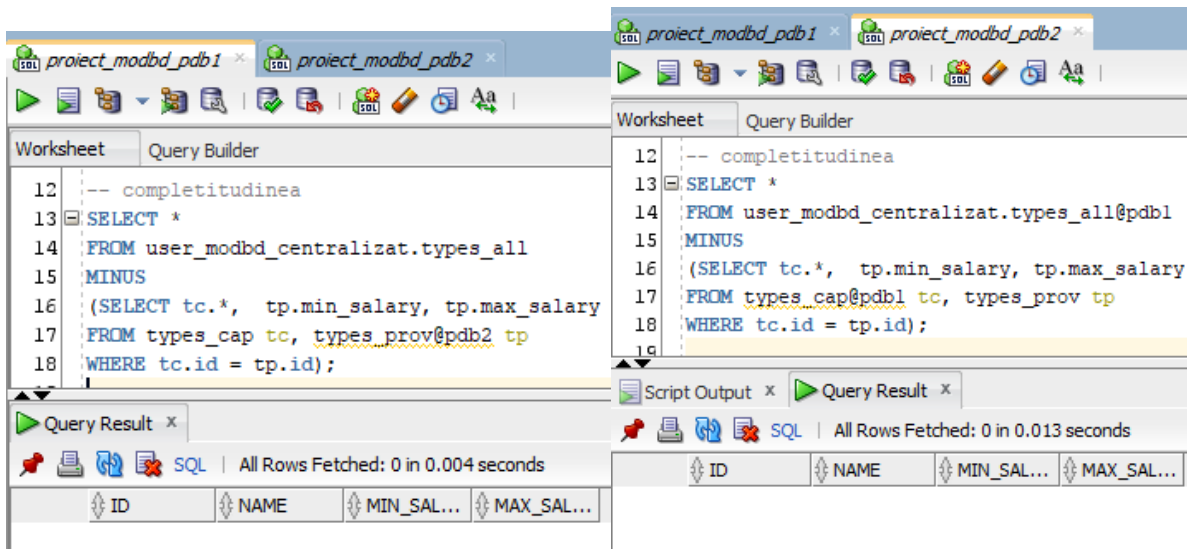
-- completitudinea
SELECT *
FROM user_modbd_centralizat.types_all
MINUS
(SELECT tc.*, tp.min_salary, tp.max_salary
FROM types_cap tc, types_prov@pdb2 tp
WHERE tc.id = tp.id);

```

```

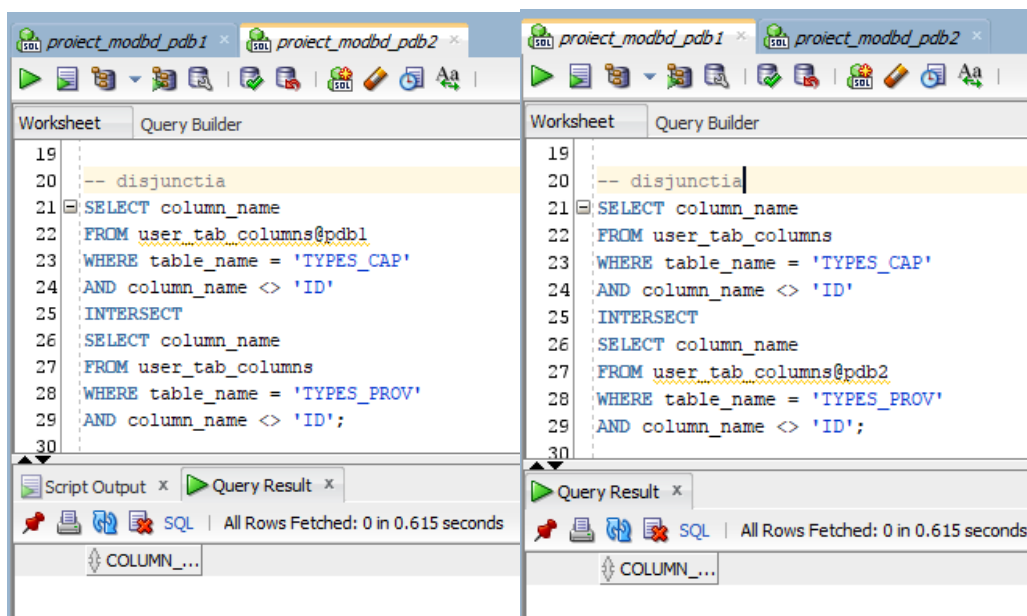
SELECT *
FROM user_modbd_centralizat.types_all@pdb1
MINUS
(SELECT tc.*, tp.min_salary, tp.max_salary
FROM types_cap@pdb1 tc, types_prov tp
WHERE tc.id = tp.id);

```



```
-- disjunctia
SELECT column_name
FROM user_tab_columns
WHERE table_name = 'TYPES_CAP'
AND column_name <> 'ID'
INTERSECT
SELECT column_name
FROM user_tab_columns@pdb2
WHERE table_name = 'TYPES_PROV'
AND column_name <> 'ID';
```

```
SELECT column_name
FROM user_tab_columns@pdb1
WHERE table_name = 'TYPES_CAP'
AND column_name <> 'ID'
INTERSECT
SELECT column_name
FROM user_tab_columns
WHERE table_name = 'TYPES_PROV'
AND column_name <> 'ID';
```





### 3. Popularea cu date a bazelor de date

```
INSERT INTO cities_all (id, name, region) VALUES (1, 'Bucuresti', 'capital');
INSERT INTO cities_all (id, name, region) VALUES (2, 'Constanta', 'province');
INSERT INTO cities_all (id, name, region) VALUES (3, 'Galati', 'province');
```

```
INSERT INTO restaurants_all (id, name, city_id) VALUES (1, 'BestFood Unirii', 1);
INSERT INTO restaurants_all (id, name, city_id) VALUES (2, 'BestFood Herastrau', 1);
INSERT INTO restaurants_all (id, name, city_id) VALUES (3, 'BestFood Tineretului', 1);
INSERT INTO restaurants_all (id, name, city_id) VALUES (4, 'BestFood Centru', 2);
INSERT INTO restaurants_all (id, name, city_id) VALUES (5, 'BestFood Mamaia', 2);
INSERT INTO restaurants_all (id, name, city_id) VALUES (6, 'BestFood Faleza', 3);
```

```
INSERT INTO types_all (id, name, min_salary, max_salary) VALUES (1, 'waiter', 1000, 4000);
INSERT INTO types_all (id, name, min_salary, max_salary) VALUES (2, 'chef', 2000, 5000);
```

```
INSERT INTO employees_all (id, name, salary, hire_date, type_id, restaurant_id) VALUES (1, 'Ion Popescu', 4000, TO_DATE('01-01-2020', 'DD-MM-YYYY'), 1, 1);
INSERT INTO employees_all (id, name, salary, hire_date, type_id, restaurant_id) VALUES (2, 'Lucian Dumitrescu', 5000, TO_DATE('01-01-2020', 'DD-MM-YYYY'), 2, 1);
INSERT INTO employees_all (id, name, salary, hire_date, type_id, restaurant_id) VALUES (3, 'Alfred Ciobanu', 3000, TO_DATE('01-01-2020', 'DD-MM-YYYY'), 1, 2);
INSERT INTO employees_all (id, name, salary, hire_date, type_id, restaurant_id) VALUES (4, 'Iulian Ifrim', 4000, TO_DATE('01-01-2020', 'DD-MM-YYYY'), 2, 2);
INSERT INTO employees_all (id, name, salary, hire_date, type_id, restaurant_id) VALUES (5, 'Vicentiu Teodorescu', 3000, TO_DATE('01-01-2020', 'DD-MM-YYYY'), 1, 3);
INSERT INTO employees_all (id, name, salary, hire_date, type_id, restaurant_id) VALUES (6, 'Aurelian Dobre', 4000, TO_DATE('01-01-2020', 'DD-MM-YYYY'), 2, 3);
INSERT INTO employees_all (id, name, salary, hire_date, type_id, restaurant_id) VALUES (7, 'Paul Dabija', 4000, TO_DATE('01-01-2020', 'DD-MM-YYYY'), 1, 4);
INSERT INTO employees_all (id, name, salary, hire_date, type_id, restaurant_id) VALUES (8, 'Iurie Georgescu', 5000, TO_DATE('01-01-2020', 'DD-MM-YYYY'), 2, 4);
INSERT INTO employees_all (id, name, salary, hire_date, type_id, restaurant_id) VALUES (9, 'Emilian Dobre', 4000, TO_DATE('01-01-2020', 'DD-MM-YYYY'), 1, 5);
INSERT INTO employees_all (id, name, salary, hire_date, type_id, restaurant_id) VALUES (10, 'Andrei Dima', 5000, TO_DATE('01-01-2020', 'DD-MM-YYYY'), 2, 5);
INSERT INTO employees_all (id, name, salary, hire_date, type_id, restaurant_id) VALUES (11, 'Florin Cristea', 1000, TO_DATE('01-01-2020', 'DD-MM-YYYY'), 1, 6);
INSERT INTO employees_all (id, name, salary, hire_date, type_id, restaurant_id) VALUES (12, 'Ionut Mocanu', 2000, TO_DATE('01-01-2020', 'DD-MM-YYYY'), 2, 6);
```

```
INSERT INTO menus_all (id, menu_date) VALUES (1, TO_DATE('01-04-2023', 'DD-MM-YYYY'));
```

```
INSERT INTO drinks_all (id, name, type, price, menu_id) VALUES (1, 'Apa', 'non-alcoholic', 5, 1);
INSERT INTO drinks_all (id, name, type, price, menu_id) VALUES (2, 'Bere', 'alcoholic', 10, 1);
INSERT INTO drinks_all (id, name, type, price, menu_id) VALUES (3, 'Vin', 'alcoholic', 15, 1);
INSERT INTO drinks_all (id, name, type, price, menu_id) VALUES (4, 'Cola', 'non-alcoholic', 10, 1);
INSERT INTO drinks_all (id, name, type, price, menu_id) VALUES (5, 'Whiskey', 'alcoholic', 20, 1);
INSERT INTO drinks_all (id, name, type, price, menu_id) VALUES (6, 'Fresh', 'non-alcoholic', 15, 1);
```

```
INSERT INTO dishes_all (id, name, price, menu_id) VALUES (1, 'Pizza', 25, 1);
INSERT INTO dishes_all (id, name, price, menu_id) VALUES (2, 'Burger', 30, 1);
INSERT INTO dishes_all (id, name, price, menu_id) VALUES (3, 'Paste', 25, 1);
INSERT INTO dishes_all (id, name, price, menu_id) VALUES (4, 'Coaste de porc', 40, 1);
INSERT INTO dishes_all (id, name, price, menu_id) VALUES (5, 'Aripioare de pui', 30, 1);
INSERT INTO dishes_all (id, name, price, menu_id) VALUES (6, 'Cartofi prajiti', 10, 1);
```

```
INSERT INTO orders_all (id, tip, waiter_id) VALUES (1, 0, 1);
```

```

INSERT INTO orders_all (id, tip, waiter_id) VALUES (2, 0, 1);
INSERT INTO orders_all (id, tip, waiter_id) VALUES (3, 0, 3);
INSERT INTO orders_all (id, tip, waiter_id) VALUES (4, 0, 7);
INSERT INTO orders_all (id, tip, waiter_id) VALUES (5, 0, 7);
INSERT INTO orders_all (id, tip, waiter_id) VALUES (6, 0, 11);

```

```

INSERT INTO orders_drinks_all (id, count, order_id, drink_id) VALUES (1, 1, 1, 1);
INSERT INTO orders_drinks_all (id, count, order_id, drink_id) VALUES (2, 1, 2, 2);
INSERT INTO orders_drinks_all (id, count, order_id, drink_id) VALUES (3, 1, 3, 3);
INSERT INTO orders_drinks_all (id, count, order_id, drink_id) VALUES (4, 1, 4, 4);
INSERT INTO orders_drinks_all (id, count, order_id, drink_id) VALUES (5, 1, 5, 5);
INSERT INTO orders_drinks_all (id, count, order_id, drink_id) VALUES (6, 1, 6, 6);

```

```

INSERT INTO chefs_orders_dishes_all (id, count, order_id, chef_id, dish_id) VALUES (1, 1, 1, 2, 1);
INSERT INTO chefs_orders_dishes_all (id, count, order_id, chef_id, dish_id) VALUES (2, 1, 2, 2, 2);
INSERT INTO chefs_orders_dishes_all (id, count, order_id, chef_id, dish_id) VALUES (3, 1, 3, 4, 3);
INSERT INTO chefs_orders_dishes_all (id, count, order_id, chef_id, dish_id) VALUES (4, 1, 4, 8, 4);
INSERT INTO chefs_orders_dishes_all (id, count, order_id, chef_id, dish_id) VALUES (5, 1, 5, 8, 5);
INSERT INTO chefs_orders_dishes_all (id, count, order_id, chef_id, dish_id) VALUES (6, 1, 6, 12, 6);

```

The screenshot shows a SQL script execution window titled "project\_modul\_centralizat". The script contains 27 lines of SQL code, primarily INSERT statements for tables: cities\_all, restaurants\_all, types\_all, and employees\_all. The script is executed successfully, as indicated by the "Script Output" pane at the bottom, which shows "Task completed in 0.313 seconds" and a list of 12 rows inserted.

```

1  INSERT INTO cities_all (id, name, region) VALUES (1, 'Bucuresti', 'capital');
2  INSERT INTO cities_all (id, name, region) VALUES (2, 'Constanta', 'province');
3  INSERT INTO cities_all (id, name, region) VALUES (3, 'Galati', 'province');
4
5  INSERT INTO restaurants_all (id, name, city_id) VALUES (1, 'BestFood Unirii', 1);
6  INSERT INTO restaurants_all (id, name, city_id) VALUES (2, 'BestFood Berceni', 1);
7  INSERT INTO restaurants_all (id, name, city_id) VALUES (3, 'BestFood Tineretului', 1);
8  INSERT INTO restaurants_all (id, name, city_id) VALUES (4, 'BestFood Centru', 2);
9  INSERT INTO restaurants_all (id, name, city_id) VALUES (5, 'BestFood Mamaia', 2);
10 INSERT INTO restaurants_all (id, name, city_id) VALUES (6, 'BestFood Faleza', 3);
11
12 INSERT INTO types_all (id, name, min_salary, max_salary) VALUES (1, 'waiter', 1000, 4000);
13 INSERT INTO types_all (id, name, min_salary, max_salary) VALUES (2, 'chef', 2000, 5000);
14
15 INSERT INTO employees_all (id, name, salary, hire_date, type_id, restaurant_id) VALUES (1, 'Ion Popescu', 4000, TO_DATE('01-01-2020', 'DD-MM-YYYY'), 1, 1);
16 INSERT INTO employees_all (id, name, salary, hire_date, type_id, restaurant_id) VALUES (2, 'Lucian Dumitrescu', 5000, TO_DATE('01-01-2020', 'DD-MM-YYYY'), 2, 1);
17 INSERT INTO employees_all (id, name, salary, hire_date, type_id, restaurant_id) VALUES (3, 'Alfred Ciobanu', 3000, TO_DATE('01-01-2020', 'DD-MM-YYYY'), 1, 2);
18 INSERT INTO employees_all (id, name, salary, hire_date, type_id, restaurant_id) VALUES (4, 'Tulian Iftim', 4000, TO_DATE('01-01-2020', 'DD-MM-YYYY'), 2, 2);
19 INSERT INTO employees_all (id, name, salary, hire_date, type_id, restaurant_id) VALUES (5, 'Vicentiu Teodorescu', 3000, TO_DATE('01-01-2020', 'DD-MM-YYYY'), 1, 3);
20 INSERT INTO employees_all (id, name, salary, hire_date, type_id, restaurant_id) VALUES (6, 'Aurelian Dobres', 4000, TO_DATE('01-01-2020', 'DD-MM-YYYY'), 2, 3);
21 INSERT INTO employees_all (id, name, salary, hire_date, type_id, restaurant_id) VALUES (7, 'Paul Dabija', 4000, TO_DATE('01-01-2020', 'DD-MM-YYYY'), 1, 4);
22 INSERT INTO employees_all (id, name, salary, hire_date, type_id, restaurant_id) VALUES (8, 'Tutiu Georgescu', 5000, TO_DATE('01-01-2020', 'DD-MM-YYYY'), 2, 4);
23 INSERT INTO employees_all (id, name, salary, hire_date, type_id, restaurant_id) VALUES (9, 'Emilian Dobres', 4000, TO_DATE('01-01-2020', 'DD-MM-YYYY'), 1, 5);
24 INSERT INTO employees_all (id, name, salary, hire_date, type_id, restaurant_id) VALUES (10, 'Andrei Dima', 5000, TO_DATE('01-01-2020', 'DD-MM-YYYY'), 2, 5);
25 INSERT INTO employees_all (id, name, salary, hire_date, type_id, restaurant_id) VALUES (11, 'Florin Cristea', 1000, TO_DATE('01-01-2020', 'DD-MM-YYYY'), 1, 6);
26 INSERT INTO employees_all (id, name, salary, hire_date, type_id, restaurant_id) VALUES (12, 'Ionut Mocanu', 2000, TO_DATE('01-01-2020', 'DD-MM-YYYY'), 2, 6);
27

```

Script Output <

Task completed in 0.313 seconds

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

```

28 INSERT INTO menus_all (id, menu_date) VALUES (1, TO_DATE('01-04-2023', 'DD-MM-YYYY'));
29
30 INSERT INTO drinks_all (id, name, type, price, menu_id) VALUES (1, 'Apa', 'non-alcoholic', 5, 1);
31 INSERT INTO drinks_all (id, name, type, price, menu_id) VALUES (2, 'Bere', 'alcoholic', 10, 1);
32 INSERT INTO drinks_all (id, name, type, price, menu_id) VALUES (3, 'Vin', 'alcoholic', 15, 1);
33 INSERT INTO drinks_all (id, name, type, price, menu_id) VALUES (4, 'Cola', 'non-alcoholic', 10, 1);
34 INSERT INTO drinks_all (id, name, type, price, menu_id) VALUES (5, 'Whiskey', 'alcoholic', 20, 1);
35 INSERT INTO drinks_all (id, name, type, price, menu_id) VALUES (6, 'Fresh', 'non-alcoholic', 15, 1);
36
37 INSERT INTO dishes_all (id, name, price, menu_id) VALUES (1, 'Pizza', 25, 1);
38 INSERT INTO dishes_all (id, name, price, menu_id) VALUES (2, 'Burger', 30, 1);
39 INSERT INTO dishes_all (id, name, price, menu_id) VALUES (3, 'Paste', 25, 1);
40 INSERT INTO dishes_all (id, name, price, menu_id) VALUES (4, 'Coaste de porc', 40, 1);
41 INSERT INTO dishes_all (id, name, price, menu_id) VALUES (5, 'Aripioare de pui', 30, 1);
42 INSERT INTO dishes_all (id, name, price, menu_id) VALUES (6, 'Cartofi prajiti', 10, 1);
43
44 INSERT INTO orders_all (id, tip, waiter_id) VALUES (1, 0, 1);
45 INSERT INTO orders_all (id, tip, waiter_id) VALUES (2, 0, 1);
46 INSERT INTO orders_all (id, tip, waiter_id) VALUES (3, 0, 3);
47 INSERT INTO orders_all (id, tip, waiter_id) VALUES (4, 0, 7);
48 INSERT INTO orders_all (id, tip, waiter_id) VALUES (5, 0, 7);
49 INSERT INTO orders_all (id, tip, waiter_id) VALUES (6, 0, 11);
50
51 INSERT INTO orders_drinks_all (id, count, order_id, drink_id) VALUES (1, 1, 1, 1);
52 INSERT INTO orders_drinks_all (id, count, order_id, drink_id) VALUES (2, 1, 2, 2);
53 INSERT INTO orders_drinks_all (id, count, order_id, drink_id) VALUES (3, 1, 3, 3);
54 INSERT INTO orders_drinks_all (id, count, order_id, drink_id) VALUES (4, 1, 4, 4);
55 INSERT INTO orders_drinks_all (id, count, order_id, drink_id) VALUES (5, 1, 5, 5);
56 INSERT INTO orders_drinks_all (id, count, order_id, drink_id) VALUES (6, 1, 6, 6);
57
58 INSERT INTO chefs_orders_dishes_all (id, count, order_id, chef_id, dish_id) VALUES (1, 1, 1, 2, 1);
59 INSERT INTO chefs_orders_dishes_all (id, count, order_id, chef_id, dish_id) VALUES (2, 1, 2, 2, 2);
60 INSERT INTO chefs_orders_dishes_all (id, count, order_id, chef_id, dish_id) VALUES (3, 1, 3, 4, 3);
61 INSERT INTO chefs_orders_dishes_all (id, count, order_id, chef_id, dish_id) VALUES (4, 1, 4, 8, 4);
62 INSERT INTO chefs_orders_dishes_all (id, count, order_id, chef_id, dish_id) VALUES (5, 1, 5, 8, 5);
63 INSERT INTO chefs_orders_dishes_all (id, count, order_id, chef_id, dish_id) VALUES (6, 1, 6, 12, 6);

```

Script Output x  
 | Task completed in 0.349 seconds

1 row inserted.

1 row inserted.

1 row inserted.

## 4. Furnizarea formelor de transparență pentru întreg modelul ales

### a. Transparență pentru fragmentele verticale

Pentru a realiza transparența fragmentelor verticale, în global, am creat un view numit types ca fiind un join al tabelelor types\_cap și types\_prov. Apoi, am definit un trigger care, atunci când se realizează o operație de insert, update sau delete asupra view-ului, modificările să aibă loc asupra tabelelor locale din bazele pdb1 și pdb2.

-- transparenta fragmentari verticale

-- types

CREATE OR REPLACE VIEW types

AS

SELECT tc.\*, tp.min\_salary, tp.max\_salary

FROM user\_modbd.types\_cap tc, user\_modbd.types\_prov@pdb2 tp

WHERE tc.id = tp.id;

CREATE OR REPLACE TRIGGER trigger\_types

INSTEAD OF INSERT OR UPDATE OR DELETE ON types

```

FOR EACH ROW
BEGIN
  IF INSERTING THEN
    INSERT INTO user_modbd.types_cap VALUES (:NEW.id, :NEW.name);
    INSERT INTO user_modbd.types_prov@pdb2 VALUES (:NEW.id, :NEW.min_salary,
:NEW.max_salary);
  ELSIF UPDATING THEN
    UPDATE user_modbd.types_cap SET id = :OLD.id, name = :NEW.name WHERE id = :OLD.id;
    UPDATE user_modbd.types_prov@pdb2 SET id = :OLD.id, min_salary = :NEW.min_salary,
max_salary = :NEW.max_salary WHERE id = :OLD.id;
  ELSIF DELETING THEN
    DELETE FROM user_modbd.types_cap WHERE id = :OLD.id;
    DELETE FROM user_modbd.types_prov@pdb2 WHERE id = :OLD.id;
  END IF;
END;
/

```

```

1  -- transparenta fragmentari verticale
2  -- types
3  CREATE OR REPLACE VIEW types
4  AS
5  SELECT tc.*, tp.min_salary, tp.max_salary
6  FROM user_modbd.types_cap tc, user_modbd.types_prov@pdb2 tp
7  WHERE tc.id = tp.id;
8
9  CREATE OR REPLACE TRIGGER trigger_types
10  INSTEAD OF INSERT OR UPDATE OR DELETE ON types
11  FOR EACH ROW
12  BEGIN
13  IF INSERTING THEN
14    INSERT INTO user_modbd.types_cap VALUES (:NEW.id, :NEW.name);
15    INSERT INTO user_modbd.types_prov@pdb2 VALUES (:NEW.id, :NEW.min_salary, :NEW.max_salary);
16  ELSIF UPDATING THEN
17    UPDATE user_modbd.types_cap SET id = :OLD.id, name = :NEW.name WHERE id = :OLD.id;
18    UPDATE user_modbd.types_prov@pdb2 SET id = :OLD.id, min_salary = :NEW.min_salary, max_salary = :NEW.max_salary WHERE id = :OLD.id;
19  ELSIF DELETING THEN
20    DELETE FROM user_modbd.types_cap WHERE id = :OLD.id;
21    DELETE FROM user_modbd.types_prov@pdb2 WHERE id = :OLD.id;
22  END IF;
23  END;
24  /
25
Script Output x
Task completed in 0.219 seconds

View TYPES created.

Trigger TRIGGER_TYPES compiled

```

## b. Transparență pentru fragmentele orizontale

Pentru transparența fragmentelor orizontale, am creat view-uri ca uniuni ale tabelor din pdb1 și pdb2, apoi am definit trigger-i care, în caz de insert, update sau delete, să efectueze modificări asupra bazelor de date aferente. De exemplu, în cazul cities, trigger-ul verifică dacă regiunea este capitală sau provincie și efectuează operațiile asupra bazei respective.

```

-- transparenta fragmentari orizontale
-- cities
CREATE OR REPLACE VIEW cities
AS
SELECT * FROM user_modbd.cities_cap
UNION ALL

```

```

SELECT * FROM user_modbd.cities_prov@pdb2;

CREATE OR REPLACE TRIGGER trigger_cities
  INSTEAD OF INSERT OR UPDATE OR DELETE ON cities
  FOR EACH ROW
BEGIN
  IF INSERTING THEN
    IF :NEW.region = 'capital' THEN
      INSERT INTO user_modbd.cities_cap VALUES (:NEW.id, :NEW.name, :NEW.region);
    ELSEIF :NEW.region = 'province' THEN
      INSERT INTO user_modbd.cities_prov@pdb2 VALUES (:NEW.id, :NEW.name, :NEW.region);
    END IF;
  ELSEIF UPDATING THEN
    IF :OLD.region = 'capital' THEN
      UPDATE user_modbd.cities_cap SET id = :OLD.id, name = :NEW.name, region = :OLD.region
WHERE id = :OLD.id;
    ELSEIF :OLD.region = 'province' THEN
      UPDATE user_modbd.cities_prov@pdb2 SET id = :OLD.id, name = :NEW.name, region =
:OLD.region WHERE id = :OLD.id;
    END IF;
  ELSEIF DELETING THEN
    IF :OLD.region = 'capital' THEN
      DELETE FROM user_modbd.cities_cap WHERE id = :OLD.id;
    ELSEIF :OLD.region = 'province' THEN
      DELETE FROM user_modbd.cities_prov@pdb2 WHERE id = :OLD.id;
    END IF;
  END IF;
END;
/

```

```

1  -- transparenta fragmentari orizontale
2  -- cities
3  CREATE OR REPLACE VIEW cities
4  AS
5  SELECT * FROM user_modbd.cities_cap
6  UNION ALL
7  SELECT * FROM user_modbd.cities_prov@pdb2;
8
9  CREATE OR REPLACE TRIGGER trigger_cities
10  INSTEAD OF INSERT OR UPDATE OR DELETE ON cities
11  FOR EACH ROW
12  BEGIN
13  IF INSERTING THEN
14  IF :NEW.region = 'capital' THEN
15      INSERT INTO user_modbd.cities_cap VALUES (:NEW.id, :NEW.name, :NEW.region);
16  ELSIF :NEW.region = 'province' THEN
17      INSERT INTO user_modbd.cities_prov@pdb2 VALUES (:NEW.id, :NEW.name, :NEW.region);
18  END IF;
19  ELSIF UPDATING THEN
20  IF :OLD.region = 'capital' THEN
21      UPDATE user_modbd.cities_cap SET id = :OLD.id, name = :NEW.name, region = :OLD.region WHERE id = :OLD.id;
22  ELSIF :OLD.region = 'province' THEN
23      UPDATE user_modbd.cities_prov@pdb2 SET id = :OLD.id, name = :NEW.name, region = :OLD.region WHERE id = :OLD.id;
24  END IF;
25  ELSIF DELETING THEN
26  IF :OLD.region = 'capital' THEN
27      DELETE FROM user_modbd.cities_cap WHERE id = :OLD.id;
28  ELSIF :OLD.region = 'province' THEN
29      DELETE FROM user_modbd.cities_prov@pdb2 WHERE id = :OLD.id;
30  END IF;
31  END IF;
32  END;

```

Script Output x

Task completed in 0.067 seconds

View CITIES created.

Trigger TRIGGER\_CITIES compiled

Pentru tabelele fragmentate orizontal derivat, trigger-i verifică în tabela părinte dacă se află cheia externă.

```

-- restaurants
CREATE OR REPLACE VIEW restaurants
AS
SELECT * FROM user_modbd.restaurants_cap
UNION ALL
SELECT * FROM user_modbd.restaurants_prov@pdb2;

CREATE OR REPLACE TRIGGER trigger_restaurants
INSTEAD OF INSERT OR UPDATE OR DELETE ON restaurants
FOR EACH ROW
DECLARE
    v_count INT;
BEGIN
    IF INSERTING THEN
        SELECT count(id) INTO v_count FROM user_modbd.cities_cap WHERE id = :NEW.city_id;
        IF (v_count <> 0) THEN
            INSERT INTO user_modbd.restaurants_cap VALUES (:NEW.id, :NEW.name, :NEW.city_id);
        END IF;
        SELECT count(id) INTO v_count FROM user_modbd.cities_prov@pdb2 WHERE id =
:NEW.city_id;
        IF (v_count <> 0) THEN

```

```

        INSERT INTO user_modbd.restaurants_prov@pdb2 VALUES (:NEW.id, :NEW.name,
:NEW.city_id);
    END IF;
    ELSIF UPDATING THEN
        SELECT count(id) INTO v_count FROM user_modbd.cities_cap WHERE id = :OLD.city_id;
        IF (v_count <> 0) THEN
            UPDATE user_modbd.restaurants_cap SET id = :OLD.id, name = :NEW.name, city_id =
:OLD.city_id WHERE id = :OLD.id;
        END IF;
        SELECT count(id) INTO v_count FROM user_modbd.cities_prov@pdb2 WHERE id =
:OLD.city_id;
        IF (v_count <> 0) THEN
            UPDATE user_modbd.restaurants_prov@pdb2 SET id = :OLD.id, name = :NEW.name, city_id
= :OLD.city_id WHERE id = :OLD.id;
        END IF;
    ELSIF DELETING THEN
        SELECT count(id) INTO v_count FROM user_modbd.cities_cap WHERE id = :OLD.city_id;
        IF (v_count <> 0) THEN
            DELETE FROM user_modbd.restaurants_cap WHERE id = :OLD.id;
        END IF;
        SELECT count(id) INTO v_count FROM user_modbd.cities_prov@pdb2 WHERE id =
:OLD.city_id;
        IF (v_count <> 0) THEN
            DELETE FROM user_modbd.restaurants_prov@pdb2 WHERE id = :OLD.id;
        END IF;
    END IF;
END;
/

```

```

4 SELECT * FROM user_modbd.restaurants_cap
5 UNION ALL
6 SELECT * FROM user_modbd.restaurants_prov@pdb2;
7
8 CREATE OR REPLACE TRIGGER trigger_restaurants
9   INSTEAD OF INSERT OR UPDATE OR DELETE ON restaurants
10  FOR EACH ROW
11  DECLARE
12    v_count INT;
13  BEGIN
14    IF INSERTING THEN
15      SELECT count(id) INTO v_count FROM user_modbd.cities_cap WHERE id = :NEW.city_id;
16      IF (v_count <> 0) THEN
17        INSERT INTO user_modbd.restaurants_cap VALUES (:NEW.id, :NEW.name, :NEW.city_id);
18      END IF;
19      SELECT count(id) INTO v_count FROM user_modbd.cities_prov@pdb2 WHERE id = :NEW.city_id;
20      IF (v_count <> 0) THEN
21        INSERT INTO user_modbd.restaurants_prov@pdb2 VALUES (:NEW.id, :NEW.name, :NEW.city_id);
22      END IF;
23    ELSIF UPDATING THEN
24      SELECT count(id) INTO v_count FROM user_modbd.cities_cap WHERE id = :OLD.city_id;
25      IF (v_count <> 0) THEN
26        UPDATE user_modbd.restaurants_cap SET id = :OLD.id, name = :NEW.name, city_id = :OLD.city_id WHERE id = :OLD.id;
27      END IF;
28      SELECT count(id) INTO v_count FROM user_modbd.cities_prov@pdb2 WHERE id = :OLD.city_id;
29      IF (v_count <> 0) THEN
30        UPDATE user_modbd.restaurants_prov@pdb2 SET id = :OLD.id, name = :NEW.name, city_id = :OLD.city_id WHERE id = :OLD.id;
31      END IF;
32    ELSIF DELETING THEN
33      SELECT count(id) INTO v_count FROM user_modbd.cities_cap WHERE id = :OLD.city_id;
34      IF (v_count <> 0) THEN
35        DELETE FROM user_modbd.restaurants_cap WHERE id = :OLD.id;
36      END IF;
37      SELECT count(id) INTO v_count FROM user_modbd.cities_prov@pdb2 WHERE id = :OLD.city_id;
38      IF (v_count <> 0) THEN
39        DELETE FROM user_modbd.restaurants_prov@pdb2 WHERE id = :OLD.id;
40      END IF;
41    END IF;
42  END;
43 /
44

```

Script Output x

Task completed in 0.092 seconds

View RESTAURANTS created.

Trigger TRIGGER\_RESTAURANTS compiled

```

-- employees
CREATE OR REPLACE VIEW employees
AS
SELECT * FROM user_modbd.employees_cap
UNION ALL
SELECT * FROM user_modbd.employees_prov@pdb2;

CREATE OR REPLACE TRIGGER trigger_employees
  INSTEAD OF INSERT OR UPDATE OR DELETE ON employees
  FOR EACH ROW
DECLARE
  v_count INT;
BEGIN
  IF INSERTING THEN
    SELECT count(id) INTO v_count FROM user_modbd.restaurants_cap WHERE id =
:NEW.restaurant_id;
    IF (v_count <> 0) THEN
      INSERT INTO user_modbd.employees_cap VALUES (:NEW.id, :NEW.name, :NEW.salary,
:NEW.hire_date, :NEW.type_id, :NEW.restaurant_id);
    END IF;
    SELECT count(id) INTO v_count FROM user_modbd.restaurants_prov@pdb2 WHERE id =
:NEW.restaurant_id;

```



```

        IF (v_count <> 0) THEN
            INSERT INTO user_modbd.employees_prov@pdb2 VALUES (:NEW.id, :NEW.name,
:NEW.salary, :NEW.hire_date, :NEW.type_id, :NEW.restaurant_id);
        END IF;
    ELSIF UPDATING THEN
        SELECT count(id) INTO v_count FROM user_modbd.restaurants_cap WHERE id =
:OLD.restaurant_id;
        IF (v_count <> 0) THEN
            UPDATE user_modbd.employees_cap SET id = :OLD.id, name = :NEW.name, salary =
:NEW.salary, hire_date = :NEW.hire_date, type_id = :NEW.type_id, restaurant_id =
:OLD.restaurant_id WHERE id = :OLD.id;
        END IF;
        SELECT count(id) INTO v_count FROM user_modbd.restaurants_prov@pdb2 WHERE id =
:OLD.restaurant_id;
        IF (v_count <> 0) THEN
            UPDATE user_modbd.employees_prov@pdb2 SET id = :OLD.id, name = :NEW.name, salary
= :NEW.salary, hire_date = :NEW.hire_date, type_id = :NEW.type_id, restaurant_id =
:OLD.restaurant_id WHERE id = :OLD.id;
        END IF;
    ELSIF DELETING THEN
        SELECT count(id) INTO v_count FROM user_modbd.restaurants_cap WHERE id =
:OLD.restaurant_id;
        IF (v_count <> 0) THEN
            DELETE FROM user_modbd.employees_cap WHERE id = :OLD.id;
        END IF;
        SELECT count(id) INTO v_count FROM user_modbd.restaurants_prov@pdb2 WHERE id =
:OLD.restaurant_id;
        IF (v_count <> 0) THEN
            DELETE FROM user_modbd.employees_prov@pdb2 WHERE id = :OLD.id;
        END IF;
    END IF;
END;
/

```

```

1  -- employees
2  @CREATE OR REPLACE VIEW employees
3  AS
4  SELECT * FROM user_modbd.employees_cap
5  UNION ALL
6  SELECT * FROM user_modbd.employees_prov@pdb2;
7
8  CREATE OR REPLACE TRIGGER trigger_employees
9  INSTEAD OF INSERT OR UPDATE OR DELETE ON employees
10 FOR EACH ROW
11 DECLARE
12     v_count INT;
13 BEGIN
14     IF INSERTING THEN
15         SELECT count(id) INTO v_count FROM user_modbd.restaurants_cap WHERE id = :NEW.restaurant_id;
16         IF (v_count <> 0) THEN
17             INSERT INTO user_modbd.employees_cap VALUES (:NEW.id, :NEW.name, :NEW.salary, :NEW.hire_date, :NEW.type_id, :NEW.restaurant_id);
18         END IF;
19         SELECT count(id) INTO v_count FROM user_modbd.restaurants_prov@pdb2 WHERE id = :NEW.restaurant_id;
20         IF (v_count <> 0) THEN
21             INSERT INTO user_modbd.employees_prov@pdb2 VALUES (:NEW.id, :NEW.name, :NEW.salary, :NEW.hire_date, :NEW.type_id, :NEW.restaurant_id);
22         END IF;
23     ELSIF UPDATING THEN
24         SELECT count(id) INTO v_count FROM user_modbd.restaurants_cap WHERE id = :OLD.restaurant_id;
25         IF (v_count <> 0) THEN
26             UPDATE user_modbd.employees_cap SET id = :OLD.id, name = :NEW.name, salary = :NEW.salary, hire_date = :NEW.hire_date, type_id = :NEW.type_id, restaurant_id = :OLD.restaurant_id WHERE id = :OLD.id;
27         END IF;
28         SELECT count(id) INTO v_count FROM user_modbd.restaurants_prov@pdb2 WHERE id = :OLD.restaurant_id;
29         IF (v_count <> 0) THEN
30             UPDATE user_modbd.employees_prov@pdb2 SET id = :OLD.id, name = :NEW.name, salary = :NEW.salary, hire_date = :NEW.hire_date, type_id = :NEW.type_id, restaurant_id = :OLD.restaurant_id WHERE id = :OLD.id;
31         END IF;
32     ELSIF DELETING THEN
33         SELECT count(id) INTO v_count FROM user_modbd.restaurants_cap WHERE id = :OLD.restaurant_id;
34         IF (v_count <> 0) THEN
35             DELETE FROM user_modbd.employees_cap WHERE id = :OLD.id;
36         END IF;
37         SELECT count(id) INTO v_count FROM user_modbd.restaurants_prov@pdb2 WHERE id = :OLD.restaurant_id;
38         IF (v_count <> 0) THEN
39             DELETE FROM user_modbd.employees_prov@pdb2 WHERE id = :OLD.id;
40         END IF;
41     END IF;
42 END;

```

Script Output

Task completed in 0.085 seconds

View EMPLOYEES created.

Trigger TRIGGER\_EMPLOYEES compiled

-- orders  
CREATE OR REPLACE VIEW orders

```

AS
SELECT * FROM user_modbd.orders_cap
UNION ALL
SELECT * FROM user_modbd.orders_prov@pdb2;

CREATE OR REPLACE TRIGGER trigger_orders
  INSTEAD OF INSERT OR UPDATE OR DELETE ON orders
  FOR EACH ROW
DECLARE
  v_count INT;
BEGIN
  IF INSERTING THEN
    SELECT count(id) INTO v_count FROM user_modbd.employees_cap WHERE id =
:NEW.waiter_id;
    IF (v_count <> 0) THEN
      INSERT INTO user_modbd.orders_cap VALUES (:NEW.id, :NEW.order_date, :NEW.total,
:NEW.tip, :NEW.waiter_id);
    END IF;
    SELECT count(id) INTO v_count FROM user_modbd.employees_prov@pdb2 WHERE id =
:NEW.waiter_id;
    IF (v_count <> 0) THEN
      INSERT INTO user_modbd.orders_prov@pdb2 VALUES (:NEW.id, :NEW.order_date,
:NEW.total, :NEW.tip, :NEW.waiter_id);
    END IF;
  ELSIF UPDATING THEN
    SELECT count(id) INTO v_count FROM user_modbd.employees_cap WHERE id =
:OLD.waiter_id;
    IF (v_count <> 0) THEN
      UPDATE user_modbd.orders_cap SET id = :OLD.id, order_date = :NEW.order_date, total =
:NEW.total, tip = :NEW.tip, waiter_id = :OLD.waiter_id WHERE id = :OLD.id;
    END IF;
    SELECT count(id) INTO v_count FROM user_modbd.employees_prov@pdb2 WHERE id =
:OLD.waiter_id;
    IF (v_count <> 0) THEN
      UPDATE user_modbd.orders_prov@pdb2 SET id = :OLD.id, order_date = :NEW.order_date,
total = :NEW.total, tip = :NEW.tip, waiter_id = :OLD.waiter_id WHERE id = :OLD.id;
    END IF;
  ELSIF DELETING THEN
    SELECT count(id) INTO v_count FROM user_modbd.employees_cap WHERE id =
:OLD.waiter_id;
    IF (v_count <> 0) THEN
      DELETE FROM user_modbd.orders_cap WHERE id = :OLD.id;
    END IF;
    SELECT count(id) INTO v_count FROM user_modbd.employees_prov@pdb2 WHERE id =
:OLD.waiter_id;
    IF (v_count <> 0) THEN
      DELETE FROM user_modbd.orders_prov@pdb2 WHERE id = :OLD.id;
    END IF;
  END IF;
END;
/

```

```

1  -- orders
2  CREATE OR REPLACE VIEW orders
3  AS
4  SELECT * FROM user_modbd.orders_cap
5  UNION ALL
6  SELECT * FROM user_modbd.orders_prov@pdb2;
7
8  CREATE OR REPLACE TRIGGER trigger_orders
9  INSTEAD OF INSERT OR UPDATE OR DELETE ON orders
10 FOR EACH ROW
11 DECLARE
12     v_count INT;
13 BEGIN
14     IF INSERTING THEN
15         SELECT count(id) INTO v_count FROM user_modbd.employees_cap WHERE id = :NEW.waiter_id;
16         IF (v_count <> 0) THEN
17             INSERT INTO user_modbd.orders_cap VALUES (:NEW.id, :NEW.order_date, :NEW.total, :NEW.tip, :NEW.waiter_id);
18         END IF;
19         SELECT count(id) INTO v_count FROM user_modbd.employees_prov@pdb2 WHERE id = :NEW.waiter_id;
20         IF (v_count <> 0) THEN
21             INSERT INTO user_modbd.orders_prov@pdb2 VALUES (:NEW.id, :NEW.order_date, :NEW.total, :NEW.tip, :NEW.waiter_id);
22         END IF;
23     ELSIF UPDATING THEN
24         SELECT count(id) INTO v_count FROM user_modbd.employees_cap WHERE id = :OLD.waiter_id;
25         IF (v_count <> 0) THEN
26             UPDATE user_modbd.orders_cap SET id = :OLD.id, order_date = :NEW.order_date, total = :NEW.total, tip = :NEW.tip, waiter_id = :OLD.waiter_id WHERE id = :OLD.id;
27         END IF;
28         SELECT count(id) INTO v_count FROM user_modbd.employees_prov@pdb2 WHERE id = :OLD.waiter_id;
29         IF (v_count <> 0) THEN
30             UPDATE user_modbd.orders_prov@pdb2 SET id = :OLD.id, order_date = :NEW.order_date, total = :NEW.total, tip = :NEW.tip, waiter_id = :OLD.waiter_id WHERE id = :OLD.id;
31         END IF;
32     ELSIF DELETING THEN
33         SELECT count(id) INTO v_count FROM user_modbd.employees_cap WHERE id = :OLD.waiter_id;
34         IF (v_count <> 0) THEN
35             DELETE FROM user_modbd.orders_cap WHERE id = :OLD.id;
36         END IF;
37         SELECT count(id) INTO v_count FROM user_modbd.employees_prov@pdb2 WHERE id = :OLD.waiter_id;
38         IF (v_count <> 0) THEN
39             DELETE FROM user_modbd.orders_prov@pdb2 WHERE id = :OLD.id;
40         END IF;

```

Script Output x

Task completed in 0.076 seconds

View ORDERS created.

Trigger TRIGGER\_ORDERS compiled

```

-- orders_drinks
CREATE OR REPLACE VIEW orders_drinks
AS
SELECT * FROM user_modbd.orders_drinks_cap
UNION ALL
SELECT * FROM user_modbd.orders_drinks_prov@pdb2;

CREATE OR REPLACE TRIGGER trigger_orders_drinks
INSTEAD OF INSERT OR UPDATE OR DELETE ON orders_drinks
FOR EACH ROW
DECLARE
    v_count INT;
BEGIN
    IF INSERTING THEN
        SELECT count(id) INTO v_count FROM user_modbd.orders_cap WHERE id = :NEW.order_id;
        IF (v_count <> 0) THEN
            INSERT INTO user_modbd.orders_drinks_cap VALUES (:NEW.id, :NEW.count, :NEW.price,
:NEW.order_id, :NEW.drink_id);
        END IF;
        SELECT count(id) INTO v_count FROM user_modbd.orders_prov@pdb2 WHERE id =
:NEW.order_id;
        IF (v_count <> 0) THEN
            INSERT INTO user_modbd.orders_drinks_prov@pdb2 VALUES (:NEW.id, :NEW.count,
:NEW.price, :NEW.order_id, :NEW.drink_id);
        END IF;
    ELSIF UPDATING THEN
        SELECT count(id) INTO v_count FROM user_modbd.orders_cap WHERE id = :OLD.order_id;
        IF (v_count <> 0) THEN
            UPDATE user_modbd.orders_drinks_cap SET id = :OLD.id, count = :NEW.count, price =
:NEW.price, order_id = :OLD.order_id, drink_id = :NEW.drink_id WHERE id = :OLD.id;
        END IF;

```

```

        SELECT count(id) INTO v_count FROM user_modbd.orders_prov@pdb2 WHERE id =
:OLD.order_id;
        IF (v_count <> 0) THEN
            UPDATE user_modbd.orders_drinks_prov@pdb2 SET id = :OLD.id, count = :NEW.count,
price = :NEW.price, order_id = :OLD.order_id, drink_id = :NEW.drink_id WHERE id = :OLD.id;
        END IF;
    ELSIF DELETING THEN
        SELECT count(id) INTO v_count FROM user_modbd.orders_cap WHERE id = :OLD.order_id;
        IF (v_count <> 0) THEN
            DELETE FROM user_modbd.orders_drinks_cap WHERE id = :OLD.id;
        END IF;
        SELECT count(id) INTO v_count FROM user_modbd.orders_prov@pdb2 WHERE id =
:OLD.order_id;
        IF (v_count <> 0) THEN
            DELETE FROM user_modbd.orders_drinks_prov@pdb2 WHERE id = :OLD.id;
        END IF;
    END IF;
END;
/

```

The screenshot shows the SQL Developer interface with a query script in the 'Query Builder' tab. The script defines a view 'orders\_drinks' and a trigger 'trigger\_orders\_drinks'. The view is created by unioning data from 'user\_modbd.orders\_drinks\_cap' and 'user\_modbd.orders\_drinks\_prov@pdb2'. The trigger is an INSTEAD OF trigger that handles INSERT, UPDATE, and DELETE operations on the view by updating the corresponding source tables. The 'Script Output' tab at the bottom shows the successful execution of the script, indicating that the view 'ORDERS\_DRINKS' was created and the trigger 'TRIGGER\_ORDERS\_DRINKS' was compiled.

```

1  -- orders_drinks
2  CREATE OR REPLACE VIEW orders_drinks
3  AS
4  SELECT * FROM user_modbd.orders_drinks_cap
5  UNION ALL
6  SELECT * FROM user_modbd.orders_drinks_prov@pdb2;
7
8  CREATE OR REPLACE TRIGGER trigger_orders_drinks
9  INSTEAD OF INSERT OR UPDATE OR DELETE ON orders_drinks
10 FOR EACH ROW
11 DECLARE
12     v_count INT;
13 BEGIN
14     IF INSERTING THEN
15         SELECT count(id) INTO v_count FROM user_modbd.orders_cap WHERE id = :NEW.order_id;
16         IF (v_count <> 0) THEN
17             INSERT INTO user_modbd.orders_drinks_cap VALUES (:NEW.id, :NEW.count, :NEW.price, :NEW.order_id, :NEW.drink_id);
18         END IF;
19         SELECT count(id) INTO v_count FROM user_modbd.orders_prov@pdb2 WHERE id = :NEW.order_id;
20         IF (v_count <> 0) THEN
21             INSERT INTO user_modbd.orders_drinks_prov@pdb2 VALUES (:NEW.id, :NEW.count, :NEW.price, :NEW.order_id, :NEW.drink_id);
22         END IF;
23     ELSIF UPDATING THEN
24         SELECT count(id) INTO v_count FROM user_modbd.orders_cap WHERE id = :OLD.order_id;
25         IF (v_count <> 0) THEN
26             UPDATE user_modbd.orders_drinks_cap SET id = :OLD.id, count = :NEW.count, price = :NEW.price, order_id = :OLD.order_id, drink_id = :NEW.drink_id WHERE id = :OLD.id;
27         END IF;
28         SELECT count(id) INTO v_count FROM user_modbd.orders_prov@pdb2 WHERE id = :OLD.order_id;
29         IF (v_count <> 0) THEN
30             UPDATE user_modbd.orders_drinks_prov@pdb2 SET id = :OLD.id, count = :NEW.count, price = :NEW.price, order_id = :OLD.order_id, drink_id = :NEW.drink_id WHERE id = :OLD.id;
31         END IF;
32     ELSIF DELETING THEN
33         SELECT count(id) INTO v_count FROM user_modbd.orders_cap WHERE id = :OLD.order_id;
34         IF (v_count <> 0) THEN
35             DELETE FROM user_modbd.orders_drinks_cap WHERE id = :OLD.id;
36         END IF;
37         SELECT count(id) INTO v_count FROM user_modbd.orders_prov@pdb2 WHERE id = :OLD.order_id;
38         IF (v_count <> 0) THEN
39             DELETE FROM user_modbd.orders_drinks_prov@pdb2 WHERE id = :OLD.id;
40         END IF;
41     END IF;
42 END;

```

Script Output x  
Task completed in 0.081 seconds

View ORDERS\_DRINKS created.

Trigger TRIGGER\_ORDERS\_DRINKS compiled

```

-- chefs_orders_dishes
CREATE OR REPLACE VIEW chefs_orders_dishes
AS
SELECT * FROM user_modbd.chefs_orders_dishes_cap
UNION ALL
SELECT * FROM user_modbd.chefs_orders_dishes_prov@pdb2;

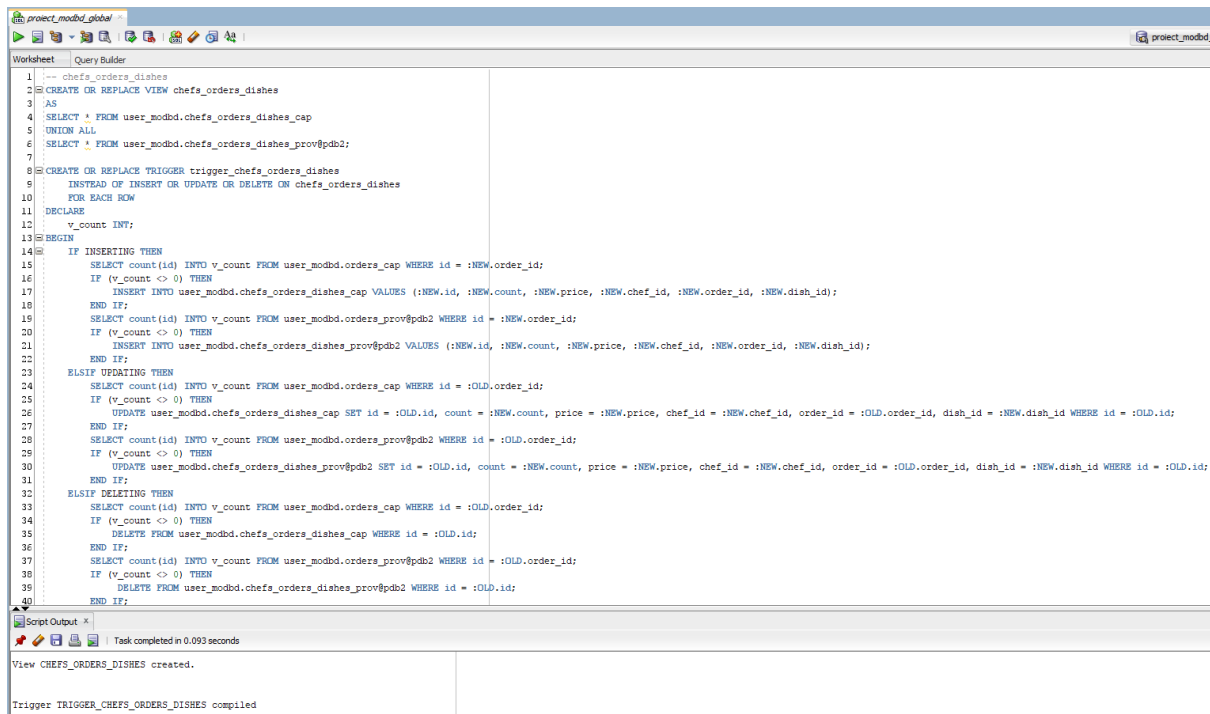
CREATE OR REPLACE TRIGGER trigger_chefs_orders_dishes
INSTEAD OF INSERT OR UPDATE OR DELETE ON chefs_orders_dishes
FOR EACH ROW
DECLARE
    v_count INT;

```

```

BEGIN
  IF INSERTING THEN
    SELECT count(id) INTO v_count FROM user_modbd.orders_cap WHERE id = :NEW.order_id;
    IF (v_count <> 0) THEN
      INSERT INTO user_modbd.chefs_orders_dishes_cap VALUES (:NEW.id, :NEW.count,
:NEW.price, :NEW.chef_id, :NEW.order_id, :NEW.dish_id);
    END IF;
    SELECT count(id) INTO v_count FROM user_modbd.orders_prov@pdb2 WHERE id =
:NEW.order_id;
    IF (v_count <> 0) THEN
      INSERT INTO user_modbd.chefs_orders_dishes_prov@pdb2 VALUES (:NEW.id,
:NEW.count, :NEW.price, :NEW.chef_id, :NEW.order_id, :NEW.dish_id);
    END IF;
  ELSIF UPDATING THEN
    SELECT count(id) INTO v_count FROM user_modbd.orders_cap WHERE id = :OLD.order_id;
    IF (v_count <> 0) THEN
      UPDATE user_modbd.chefs_orders_dishes_cap SET id = :OLD.id, count = :NEW.count, price
= :NEW.price, chef_id = :NEW.chef_id, order_id = :OLD.order_id, dish_id = :NEW.dish_id WHERE id
= :OLD.id;
    END IF;
    SELECT count(id) INTO v_count FROM user_modbd.orders_prov@pdb2 WHERE id =
:OLD.order_id;
    IF (v_count <> 0) THEN
      UPDATE user_modbd.chefs_orders_dishes_prov@pdb2 SET id = :OLD.id, count =
:NEW.count, price = :NEW.price, chef_id = :NEW.chef_id, order_id = :OLD.order_id, dish_id =
:NEW.dish_id WHERE id = :OLD.id;
    END IF;
  ELSIF DELETING THEN
    SELECT count(id) INTO v_count FROM user_modbd.orders_cap WHERE id = :OLD.order_id;
    IF (v_count <> 0) THEN
      DELETE FROM user_modbd.chefs_orders_dishes_cap WHERE id = :OLD.id;
    END IF;
    SELECT count(id) INTO v_count FROM user_modbd.orders_prov@pdb2 WHERE id =
:OLD.order_id;
    IF (v_count <> 0) THEN
      DELETE FROM user_modbd.chefs_orders_dishes_prov@pdb2 WHERE id = :OLD.id;
    END IF;
  END IF;
END;
/

```



```
1 -- chefs_orders_dishes
2 CREATE OR REPLACE VIEW chefs_orders_dishes
3 AS
4 SELECT * FROM user_modbd.chefs_orders_dishes_cap
5 UNION ALL
6 SELECT * FROM user_modbd.chefs_orders_dishes_prov@pdb2;
7
8 CREATE OR REPLACE TRIGGER trigger_chefs_orders_dishes
9 INSTEAD OF INSERT OR UPDATE OR DELETE ON chefs_orders_dishes
10 FOR EACH ROW
11 DECLARE
12     v_count INT;
13 BEGIN
14     IF INSERTING THEN
15         SELECT count(id) INTO v_count FROM user_modbd.orders_cap WHERE id = :NEW.order_id;
16         IF (v_count < 0) THEN
17             INSERT INTO user_modbd.chefs_orders_dishes_cap VALUES (:NEW.id, :NEW.count, :NEW.price, :NEW.chef_id, :NEW.order_id, :NEW.dish_id);
18         END IF;
19         SELECT count(id) INTO v_count FROM user_modbd.orders_prov@pdb2 WHERE id = :NEW.order_id;
20         IF (v_count < 0) THEN
21             INSERT INTO user_modbd.chefs_orders_dishes_prov@pdb2 VALUES (:NEW.id, :NEW.count, :NEW.price, :NEW.chef_id, :NEW.order_id, :NEW.dish_id);
22         END IF;
23     ELSIF UPDATING THEN
24         SELECT count(id) INTO v_count FROM user_modbd.orders_cap WHERE id = :OLD.order_id;
25         IF (v_count < 0) THEN
26             UPDATE user_modbd.chefs_orders_dishes_cap SET id = :OLD.id, count = :NEW.count, price = :NEW.price, chef_id = :NEW.chef_id, order_id = :OLD.order_id, dish_id = :NEW.dish_id WHERE id = :OLD.id;
27         END IF;
28         SELECT count(id) INTO v_count FROM user_modbd.orders_prov@pdb2 WHERE id = :OLD.order_id;
29         IF (v_count < 0) THEN
30             UPDATE user_modbd.chefs_orders_dishes_prov@pdb2 SET id = :OLD.id, count = :NEW.count, price = :NEW.price, chef_id = :NEW.chef_id, order_id = :OLD.order_id, dish_id = :NEW.dish_id WHERE id = :OLD.id;
31         END IF;
32     ELSIF DELETING THEN
33         SELECT count(id) INTO v_count FROM user_modbd.orders_cap WHERE id = :OLD.order_id;
34         IF (v_count < 0) THEN
35             DELETE FROM user_modbd.chefs_orders_dishes_cap WHERE id = :OLD.id;
36         END IF;
37         SELECT count(id) INTO v_count FROM user_modbd.orders_prov@pdb2 WHERE id = :OLD.order_id;
38         IF (v_count < 0) THEN
39             DELETE FROM user_modbd.chefs_orders_dishes_prov@pdb2 WHERE id = :OLD.id;
40         END IF;
41 END;
```

Script Output: x

Task completed in 0.093 seconds

View CHEFS\_ORDERS\_DISHES created.

Trigger TRIGGER\_CHEFS\_ORDERS\_DISHES compiled

### c. Transparență pentru tabelele stocate în altă bază de date față de cea la care se conectează aplicația

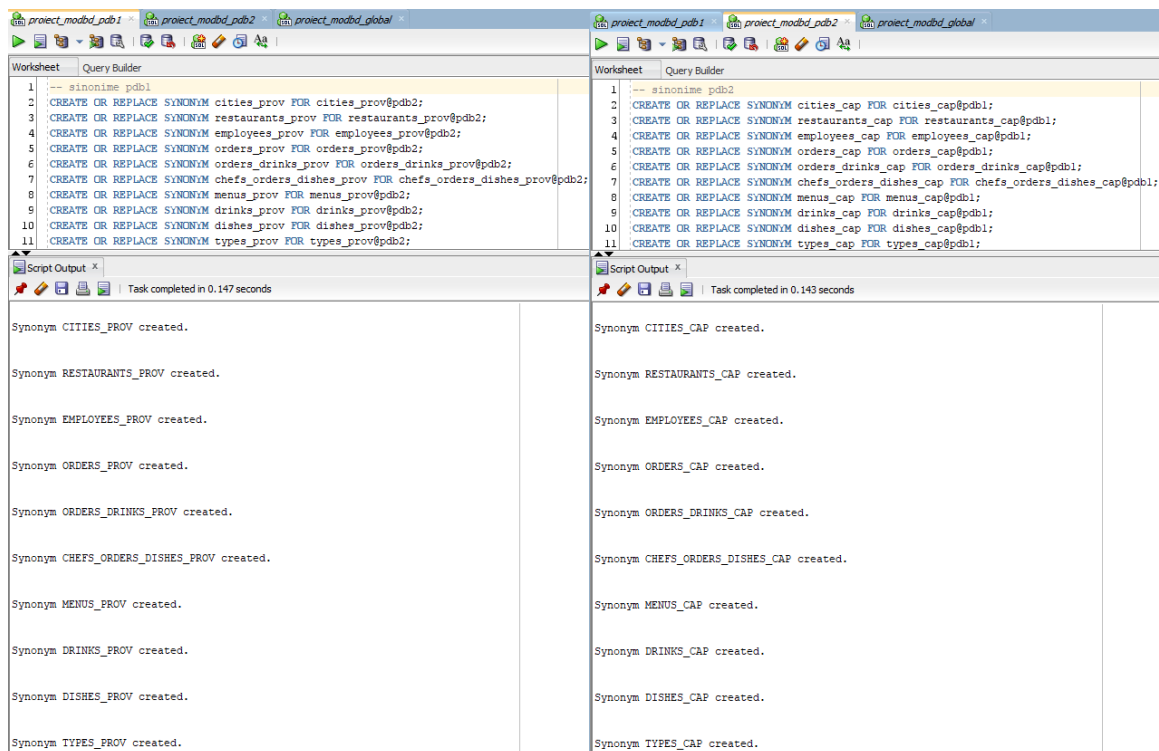
Pentru transparența tabelelor stocate în altă bază de date, am create sinonime atât în pdb1 și pdb2, cât și în global.

-- sinonime pdb1

```
CREATE OR REPLACE SYNONYM cities_prov FOR cities_prov@pdb2;
CREATE OR REPLACE SYNONYM restaurants_prov FOR restaurants_prov@pdb2;
CREATE OR REPLACE SYNONYM employees_prov FOR employees_prov@pdb2;
CREATE OR REPLACE SYNONYM orders_prov FOR orders_prov@pdb2;
CREATE OR REPLACE SYNONYM orders_drinks_prov FOR orders_drinks_prov@pdb2;
CREATE OR REPLACE SYNONYM chefs_orders_dishes_prov FOR
chefs_orders_dishes_prov@pdb2;
CREATE OR REPLACE SYNONYM menus_prov FOR menus_prov@pdb2;
CREATE OR REPLACE SYNONYM drinks_prov FOR drinks_prov@pdb2;
CREATE OR REPLACE SYNONYM dishes_prov FOR dishes_prov@pdb2;
CREATE OR REPLACE SYNONYM types_prov FOR types_prov@pdb2;
```

-- sinonime pdb2

```
CREATE OR REPLACE SYNONYM cities_cap FOR cities_cap@pdb1;
CREATE OR REPLACE SYNONYM restaurants_cap FOR restaurants_cap@pdb1;
CREATE OR REPLACE SYNONYM employees_cap FOR employees_cap@pdb1;
CREATE OR REPLACE SYNONYM orders_cap FOR orders_cap@pdb1;
CREATE OR REPLACE SYNONYM orders_drinks_cap FOR orders_drinks_cap@pdb1;
CREATE OR REPLACE SYNONYM chefs_orders_dishes_cap FOR
chefs_orders_dishes_cap@pdb1;
CREATE OR REPLACE SYNONYM menus_cap FOR menus_cap@pdb1;
CREATE OR REPLACE SYNONYM drinks_cap FOR drinks_cap@pdb1;
CREATE OR REPLACE SYNONYM dishes_cap FOR dishes_cap@pdb1;
CREATE OR REPLACE SYNONYM types_cap FOR types_cap@pdb1;
```



-- sinonime global

```

CREATE OR REPLACE SYNONYM cities_cap FOR user_modbd.cities_cap;
CREATE OR REPLACE SYNONYM cities_prov FOR user_modbd.cities_prov@pdb2;
CREATE OR REPLACE SYNONYM restaurants_cap FOR user_modbd.restaurants_cap;
CREATE OR REPLACE SYNONYM restaurants_prov FOR user_modbd.restaurants_prov@pdb2;
CREATE OR REPLACE SYNONYM employees_cap FOR user_modbd.employees_cap;
CREATE OR REPLACE SYNONYM employees_prov FOR user_modbd.employees_prov@pdb2;
CREATE OR REPLACE SYNONYM orders_cap FOR user_modbd.orders_cap;
CREATE OR REPLACE SYNONYM orders_prov FOR user_modbd.orders_prov@pdb2;
CREATE OR REPLACE SYNONYM orders_drinks_cap FOR user_modbd.orders_drinks_cap;
CREATE OR REPLACE SYNONYM orders_drinks_prov FOR
user_modbd.orders_drinks_prov@pdb2;
CREATE OR REPLACE SYNONYM chefs_orders_dishes_cap FOR
user_modbd.chefs_orders_dishes_cap;
CREATE OR REPLACE SYNONYM chefs_orders_dishes_prov FOR
user_modbd.chefs_orders_dishes_prov@pdb2;
CREATE OR REPLACE SYNONYM menus_cap FOR user_modbd.menus_cap;
CREATE OR REPLACE SYNONYM menus_prov FOR user_modbd.menus_prov@pdb2;
CREATE OR REPLACE SYNONYM drinks_cap FOR user_modbd.drinks_cap;
CREATE OR REPLACE SYNONYM drinks_prov FOR user_modbd.drinks_prov@pdb2;
CREATE OR REPLACE SYNONYM dishes_cap FOR user_modbd.dishes_cap;
CREATE OR REPLACE SYNONYM dishes_prov FOR user_modbd.dishes_prov@pdb2;
CREATE OR REPLACE SYNONYM types_cap FOR user_modbd.types_cap;
CREATE OR REPLACE SYNONYM types_prov FOR user_modbd.types_prov@pdb2;

```

project\_modbd\_pdb1 project\_modbd\_pdb2 project\_modbd\_global

Worksheet Query Builder

```

1  -- sinonime global
2  CREATE OR REPLACE SYNONYM cities_cap FOR user_modbd.cities_cap;
3  CREATE OR REPLACE SYNONYM cities_prov FOR user_modbd.cities_prov@pdb2;
4  CREATE OR REPLACE SYNONYM restaurants_cap FOR user_modbd.restaurants_cap;
5  CREATE OR REPLACE SYNONYM restaurants_prov FOR user_modbd.restaurants_prov@pdb2;
6  CREATE OR REPLACE SYNONYM employees_cap FOR user_modbd.employees_cap;
7  CREATE OR REPLACE SYNONYM employees_prov FOR user_modbd.employees_prov@pdb2;
8  CREATE OR REPLACE SYNONYM orders_cap FOR user_modbd.orders_cap;
9  CREATE OR REPLACE SYNONYM orders_prov FOR user_modbd.orders_prov@pdb2;
10 CREATE OR REPLACE SYNONYM orders_drinks_cap FOR user_modbd.orders_drinks_cap;
11 CREATE OR REPLACE SYNONYM orders_drinks_prov FOR user_modbd.orders_drinks_prov@pdb2;
12 CREATE OR REPLACE SYNONYM chefs_orders_dishes_cap FOR user_modbd.chefs_orders_dishes_cap;
13 CREATE OR REPLACE SYNONYM chefs_orders_dishes_prov FOR user_modbd.chefs_orders_dishes_prov@pdb2;
14 CREATE OR REPLACE SYNONYM menus_cap FOR user_modbd.menus_cap;
15 CREATE OR REPLACE SYNONYM menus_prov FOR user_modbd.menus_prov@pdb2;
16 CREATE OR REPLACE SYNONYM drinks_cap FOR user_modbd.drinks_cap;
17 CREATE OR REPLACE SYNONYM drinks_prov FOR user_modbd.drinks_prov@pdb2;
18 CREATE OR REPLACE SYNONYM dishes_cap FOR user_modbd.dishes_cap;
19 CREATE OR REPLACE SYNONYM dishes_prov FOR user_modbd.dishes_prov@pdb2;
20 CREATE OR REPLACE SYNONYM types_cap FOR user_modbd.types_cap;
21 CREATE OR REPLACE SYNONYM types_prov FOR user_modbd.types_prov@pdb2;

```

Script Output x

Task completed in 0.267 seconds

Synonym CITIES\_CAP created.

Synonym CITIES\_PROV created.

Synonym RESTAURANTS\_CAP created.

Synonym RESTAURANTS\_PROV created.

Synonym EMPLOYEES\_CAP created.

Synonym EMPLOYEES\_PROV created.

Synonym ORDERS\_CAP created.

Synonym ORDERS\_PROV created.

Script Output x	Script Output x
Task completed in 0.267 seconds	Task completed in 0.267 seconds
Synonym ORDERS_DRINKS_CAP created.	Synonym MENUS_CAP created.
Synonym ORDERS_DRINKS_PROV created.	Synonym MENUS_PROV created.
Synonym CHEFS_ORDERS_DISHES_CAP created.	Synonym DRINKS_CAP created.
Synonym CHEFS_ORDERS_DISHES_PROV created.	Synonym DRINKS_PROV created.
Synonym MENUS_CAP created.	Synonym DISHES_CAP created.
Synonym MENUS_PROV created.	Synonym DISHES_PROV created.
Synonym DRINKS_CAP created.	Synonym TYPES_CAP created.
Synonym DRINKS_PROV created.	Synonym TYPES_PROV created.



## 5. Asigurarea sincronizării datelor pentru relațiile replicate

Tabele menus, drinks și dishes nu depind de oraș, deoarece toate restaurantele din rețea au aceleași produse. Astfel, datele din acestea vor fi replicate. De ex, menus\_cap și menus\_prov vor conține aceleași înregistrări.

```
-- replicare pdb1
-- menus
CREATE TABLE menus_cap
AS
SELECT * FROM user_modbd_centralizat.menus_all;

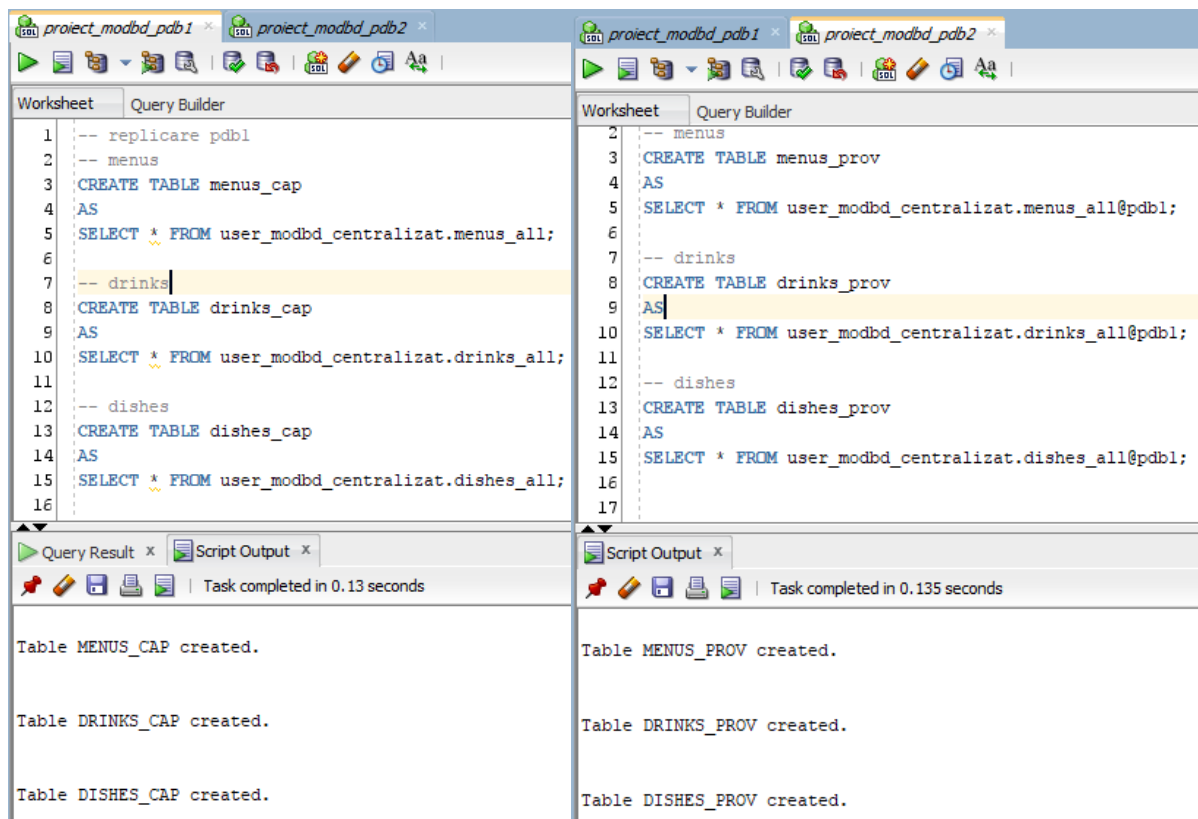
-- drinks
CREATE TABLE drinks_cap
AS
SELECT * FROM user_modbd_centralizat.drinks_all;

-- dishes
CREATE TABLE dishes_cap
AS
SELECT * FROM user_modbd_centralizat.dishes_all;

-- replicare pdb2
-- menus
CREATE TABLE menus_prov
AS
SELECT * FROM user_modbd_centralizat.menus_all@pdb1;

-- drinks
CREATE TABLE drinks_prov
AS
SELECT * FROM user_modbd_centralizat.drinks_all@pdb1;

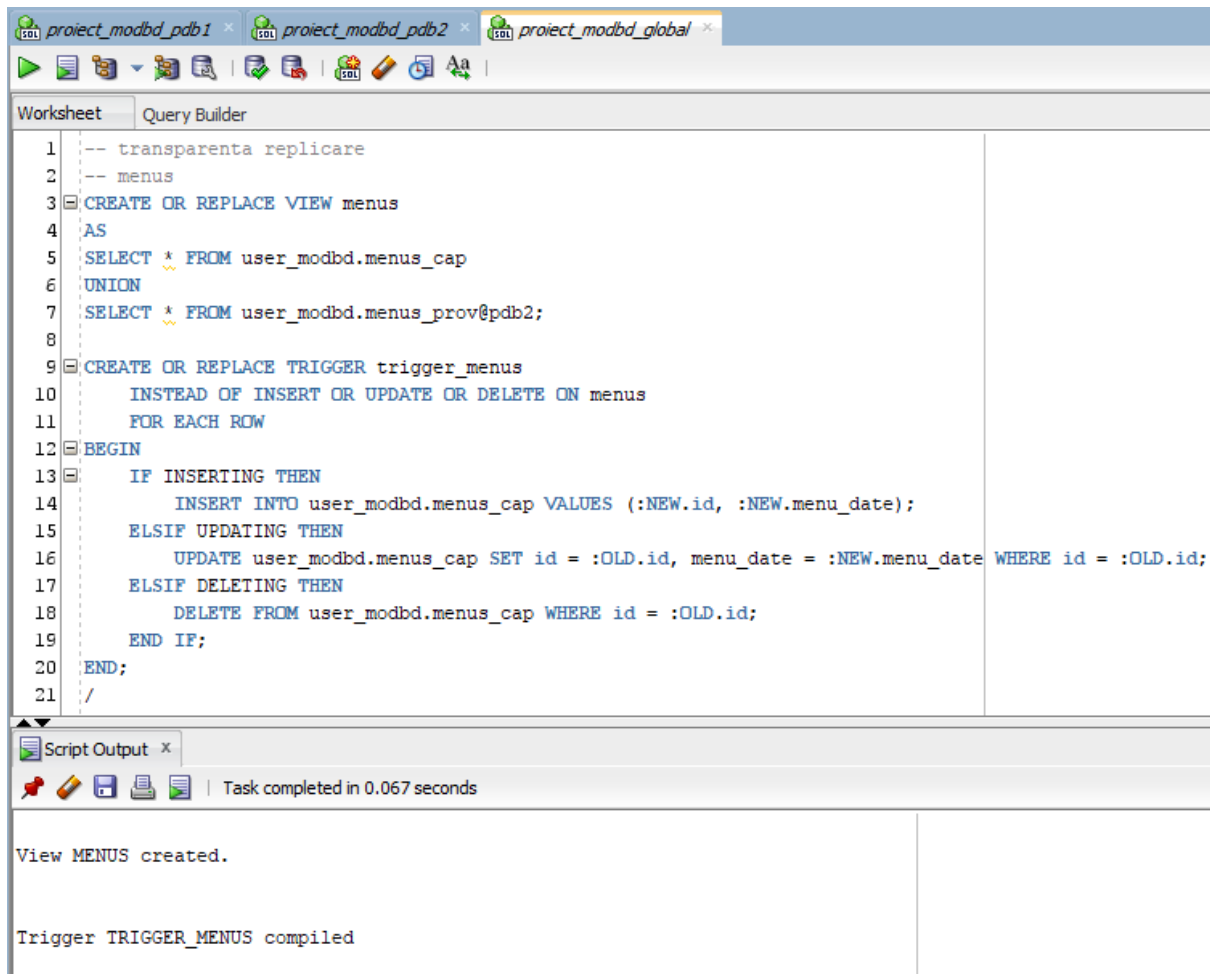
-- dishes
CREATE TABLE dishes_prov
AS
SELECT * FROM user_modbd_centralizat.dishes_all@pdb1;
```



Pentru transparență, triggerele pentru insert, update și delete se vor conecta numai la tabelele din pdb1, umând ca datele să fie copiate în pdb2.

```
-- transparenta replicare
-- menus
CREATE OR REPLACE VIEW menus
AS
SELECT * FROM user_modbd.menus_cap
UNION
SELECT * FROM user_modbd.menus_prov@pdb2;

CREATE OR REPLACE TRIGGER trigger_menus
  INSTEAD OF INSERT OR UPDATE OR DELETE ON menus
  FOR EACH ROW
BEGIN
  IF INSERTING THEN
    INSERT INTO user_modbd.menus_cap VALUES (:NEW.id, :NEW.menu_date);
  ELSIF UPDATING THEN
    UPDATE user_modbd.menus_cap SET id = :OLD.id, menu_date = :NEW.menu_date WHERE id
= :OLD.id;
  ELSIF DELETING THEN
    DELETE FROM user_modbd.menus_cap WHERE id = :OLD.id;
  END IF;
END;
/
```



```

-- drinks
CREATE OR REPLACE VIEW drinks
AS
SELECT * FROM user_modbd.drinks_cap
UNION
SELECT * FROM user_modbd.drinks_prov@pdb2;

CREATE OR REPLACE TRIGGER trigger_drinks
    INSTEAD OF INSERT OR UPDATE OR DELETE ON drinks
    FOR EACH ROW
BEGIN
    IF INSERTING THEN
        INSERT INTO user_modbd.drinks_cap VALUES (:NEW.id, :NEW.name, :NEW.type, :NEW.price,
:NEW.menu_id);
    ELSIF UPDATING THEN
        UPDATE user_modbd.drinks_cap SET id = :OLD.id, name = :NEW.name, type = :NEW.type,
price = :NEW.price, menu_id = :OLD.menu_id WHERE id = :OLD.id;
    ELSIF DELETING THEN
        DELETE FROM user_modbd.drinks_cap WHERE id = :OLD.id;
    END IF;
END;
/

```

The screenshot shows the Oracle SQL Developer interface. The top pane, titled 'Worksheet', contains a SQL script for creating a view and a trigger. The script is as follows:

```
1  -- drinks
2  CREATE OR REPLACE VIEW drinks
3  AS
4  SELECT * FROM user_modbd.drinks_cap
5  UNION
6  SELECT * FROM user_modbd.drinks_prov@pdb2;
7
8  CREATE OR REPLACE TRIGGER trigger_drinks
9  INSTEAD OF INSERT OR UPDATE OR DELETE ON drinks
10 FOR EACH ROW
11 BEGIN
12 IF INSERTING THEN
13   INSERT INTO user_modbd.drinks_cap VALUES (:NEW.id, :NEW.name, :NEW.price, :NEW.menu_id, :NEW.type);
14 ELSIF UPDATING THEN
15   UPDATE user_modbd.drinks_cap SET id = :OLD.id, name = :NEW.name, price = :NEW.price, menu_id = :OLD.menu_id, type = :NEW.type WHERE id = :OLD.id;
16 ELSIF DELETING THEN
17   DELETE FROM user_modbd.drinks_cap WHERE id = :OLD.id;
18 END IF;
19 END;
20 /
```

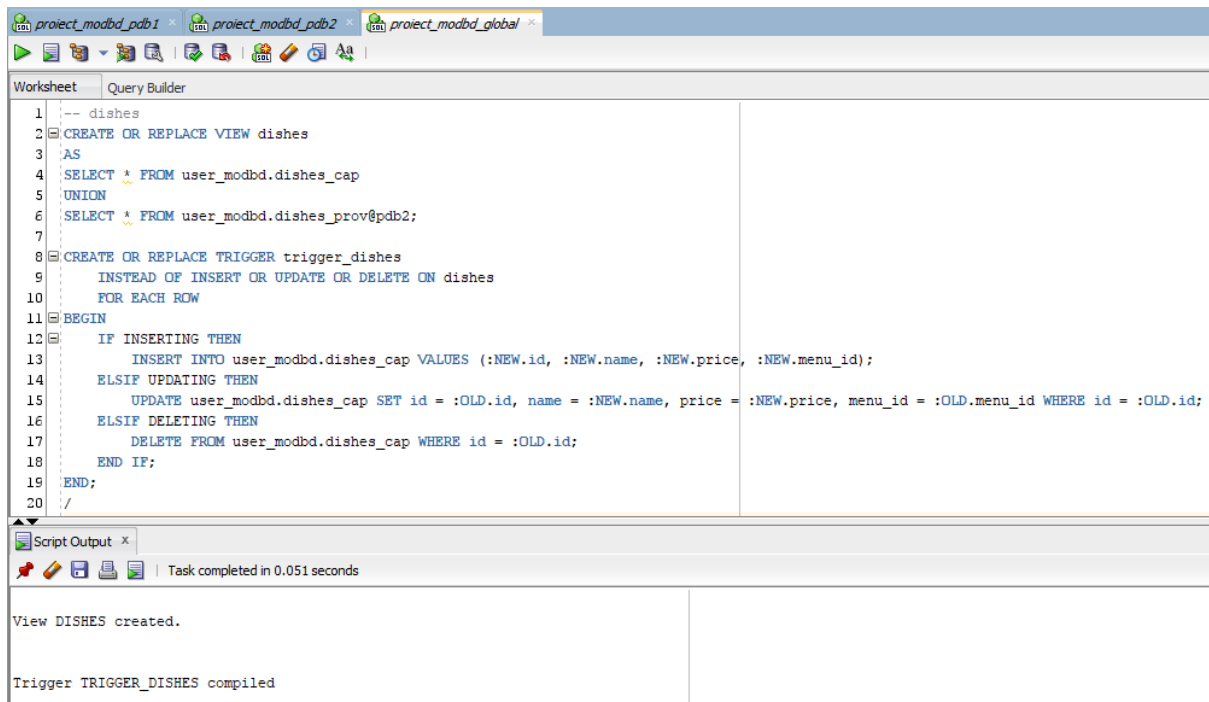
The bottom pane, titled 'Script Output', shows the results of the script execution:

```
View DRINKS created.

Trigger TRIGGER_DRINKS compiled
```

```
-- dishes
CREATE OR REPLACE VIEW dishes
AS
SELECT * FROM user_modbd.dishes_cap
UNION
SELECT * FROM user_modbd.dishes_prov@pdb2;

CREATE OR REPLACE TRIGGER trigger_dishes
  INSTEAD OF INSERT OR UPDATE OR DELETE ON dishes
  FOR EACH ROW
BEGIN
  IF INSERTING THEN
    INSERT INTO user_modbd.dishes_cap VALUES (:NEW.id, :NEW.name, :NEW.price,
:NEW.menu_id);
  ELSIF UPDATING THEN
    UPDATE user_modbd.dishes_cap SET id = :OLD.id, name = :NEW.name, price = :NEW.price,
menu_id = :OLD.menu_id WHERE id = :OLD.id;
  ELSIF DELETING THEN
    DELETE FROM user_modbd.dishes_cap WHERE id = :OLD.id;
  END IF;
END;
/
```



Pentru a sincroniza datele din pdb1 în pdb2, au fost creați trigger-i care să copieze rezultatul fiecărei operații de insert, update sau delete.

-- replicare pdb1

-- menus

```
CREATE OR REPLACE TRIGGER trigger_menus_cap
  AFTER INSERT OR UPDATE OR DELETE ON menus_cap
  FOR EACH ROW
BEGIN
  IF INSERTING THEN
    INSERT INTO menus_prov@pdb2 VALUES (:NEW.id, :NEW.menu_date);
  ELSIF UPDATING THEN
    UPDATE menus_prov@pdb2 SET id = :OLD.id, menu_date = :NEW.menu_date WHERE id =
:OLD.id;
  ELSIF DELETING THEN
    DELETE FROM menus_prov@pdb2 WHERE id = :OLD.id;
  END IF;
END;
/
```

-- drinks

```
CREATE OR REPLACE TRIGGER trigger_drinks_cap
  AFTER INSERT OR UPDATE OR DELETE ON drinks_cap
  FOR EACH ROW
BEGIN
  IF INSERTING THEN
    INSERT INTO drinks_prov@pdb2 VALUES (:NEW.id, :NEW.name, :NEW.type, :NEW.price,
:NEW.menu_id);
  ELSIF UPDATING THEN
    UPDATE drinks_prov@pdb2 SET id = :OLD.id, name = :NEW.name, type = :NEW.type, price =
:NEW.price, menu_id = :OLD.menu_id WHERE id = :OLD.id;
  ELSIF DELETING THEN
    DELETE FROM drinks_prov@pdb2 WHERE id = :OLD.id;
  END IF;
END;
/
```

```

-- dishes
CREATE OR REPLACE TRIGGER trigger_dishes_cap
  AFTER INSERT OR UPDATE OR DELETE ON dishes_cap
  FOR EACH ROW
BEGIN
  IF INSERTING THEN
    INSERT INTO dishes_prov@pdb2 VALUES (:NEW.id, :NEW.name, :NEW.price, :NEW.menu_id);
  ELSIF UPDATING THEN
    UPDATE dishes_prov@pdb2 SET id = :OLD.id, name = :NEW.name, price = :NEW.price,
menu_id = :OLD.menu_id WHERE id = :OLD.id;
  ELSIF DELETING THEN
    DELETE FROM dishes_prov@pdb2 WHERE id = :OLD.id;
  END IF;
END;
/

```

```

3 CREATE OR REPLACE TRIGGER trigger_menus_cap
4   AFTER INSERT OR UPDATE OR DELETE ON menus_cap
5   FOR EACH ROW
6 BEGIN
7   IF INSERTING THEN
8     INSERT INTO menus_prov@pdb2 VALUES (:NEW.id, :NEW.menu_date);
9   ELSIF UPDATING THEN
10    UPDATE menus_prov@pdb2 SET id = :OLD.id, menu_date = :NEW.menu_date WHERE id = :OLD.id;
11  ELSIF DELETING THEN
12    DELETE FROM menus_prov@pdb2 WHERE id = :OLD.id;
13  END IF;
14 END;
15 /
16
17 -- drinks
18 CREATE OR REPLACE TRIGGER trigger_drinks_cap
19   AFTER INSERT OR UPDATE OR DELETE ON drinks_cap
20   FOR EACH ROW
21 BEGIN
22   IF INSERTING THEN
23     INSERT INTO drinks_prov@pdb2 VALUES (:NEW.id, :NEW.name, :NEW.price, :NEW.menu_id, :NEW.type);
24   ELSIF UPDATING THEN
25     UPDATE drinks_prov@pdb2 SET id = :OLD.id, name = :NEW.name, price = :NEW.price, menu_id = :OLD.menu_id, type = :NEW.type WHERE id = :OLD.id;
26   ELSIF DELETING THEN
27     DELETE FROM drinks_prov@pdb2 WHERE id = :OLD.id;
28   END IF;
29 END;
30 /
31
32 -- dishes
33 CREATE OR REPLACE TRIGGER trigger_dishes_cap
34   AFTER INSERT OR UPDATE OR DELETE ON dishes_cap
35   FOR EACH ROW
36 BEGIN
37   IF INSERTING THEN
38     INSERT INTO dishes_prov@pdb2 VALUES (:NEW.id, :NEW.name, :NEW.price, :NEW.menu_id);
39   ELSIF UPDATING THEN
40     UPDATE dishes_prov@pdb2 SET id = :OLD.id, name = :NEW.name, price = :NEW.price, menu_id = :OLD.menu_id WHERE id = :OLD.id;
41   ELSIF DELETING THEN
42     DELETE FROM dishes_prov@pdb2 WHERE id = :OLD.id;
43   END IF;
44 END;
45 /

```

Script Output x

Task completed in 0.075 seconds

Trigger TRIGGER\_MENUS\_CAP compiled

Trigger TRIGGER\_DRINKS\_CAP compiled

Trigger TRIGGER\_DISHES\_CAP compiled

## 6. Asigurarea tuturor constrângerilor de integritate folosite în model

Pentru definirea constrângerilor, am adăugat constrângerile de cheie primară și de chei externe pentru fiecare tabel din pdb1 și pdb2. Apoi, am creat câte o secvență care să genereze

automat id-uri noi. Pentru tabelele replicate orizontal, înregistrările din pdb1 vor avea valori pare, iar cele din pdb2 vor avea valori impare. Am creat trigger-e care să adauge automat câte un nou id folosind secvența creată. În plus, avem câte un trigger în cazul tabelelor fragmentate orizontal care să verifice unicitatea cheii primare la nivel global: dacă id-ul din pdb1 nu există deja în pdb2 și invers.

```
-- cities
ALTER TABLE cities_cap ADD CONSTRAINT pk_cities_cap primary key (id);

CREATE SEQUENCE sequence_cities_cap
  INCREMENT BY 2
  START WITH 100
  NOCYCLE;

CREATE OR REPLACE TRIGGER trigger_sequence_cities_cap
  BEFORE INSERT ON cities_cap
  FOR EACH ROW
BEGIN
  SELECT sequence_cities_cap.NEXTVAL INTO :NEW.id FROM dual;
END;
/

CREATE OR REPLACE TRIGGER trigger_pk_cities_cap
  BEFORE INSERT ON cities_cap
  FOR EACH ROW
DECLARE
  v_count INT;
BEGIN
  SELECT count(*) INTO v_count FROM cities_prov@pdb2 WHERE id = :NEW.id;
  IF v_count <> 0 THEN
    RAISE_APPLICATION_ERROR (-20001, 'unique constraint
(USER_MODBD.PK_CITIES_PROV) violated');
  END IF;
END;
/
```

The screenshot shows the Oracle SQL Developer interface. The top tab bar has three tabs: 'project\_modbd\_pdb1', 'project\_modbd\_pdb2', and 'project\_modbd\_global'. The 'Query Builder' tab is active, displaying a SQL script. The script includes comments and SQL commands for altering a table, creating a sequence, and creating two triggers. The 'Script Output' tab at the bottom shows the execution results, indicating that the table was altered, the sequence was created, and both triggers were compiled successfully. The task completed in 0.068 seconds.

```
1  -- cities
2  ALTER TABLE cities_cap ADD CONSTRAINT pk_cities_cap primary key (id);
3
4  CREATE SEQUENCE sequence_cities_cap
5      INCREMENT BY 2
6      START WITH 100
7      NOCYCLE;
8
9  CREATE OR REPLACE TRIGGER trigger_sequence_cities_cap
10     BEFORE INSERT ON cities_cap
11     FOR EACH ROW
12     BEGIN
13         SELECT sequence_cities_cap.NEXTVAL INTO :NEW.id FROM dual;
14     END;
15 /
16
17 CREATE OR REPLACE TRIGGER trigger_pk_cities_cap
18     BEFORE INSERT ON cities_cap
19     FOR EACH ROW
20     DECLARE
21         v_count INT;
22     BEGIN
23         SELECT count(*) INTO v_count FROM cities_prov@pdb2 WHERE id = :NEW.id;
24         IF v_count <> 0 THEN
25             RAISE_APPLICATION_ERROR (-20001, 'unique constraint (USER_MODBD.PK_CITIES_PROV) violated');
26         END IF;
27     END;
28 /
```

Script Output x

Task completed in 0.068 seconds

Table CITIES\_CAP altered.

Sequence SEQUENCE\_CITIES\_CAP created.

Trigger TRIGGER\_SEQUENCE\_CITIES\_CAP compiled

Trigger TRIGGER\_PK\_CITIES\_CAP compiled

```
-- cities
ALTER TABLE cities_prov ADD CONSTRAINT pk_cities_prov primary key (id);

CREATE SEQUENCE sequence_cities_prov
    INCREMENT BY 2
    START WITH 101
    NOCYCLE;

CREATE OR REPLACE TRIGGER trigger_sequence_cities_prov
    BEFORE INSERT ON cities_prov
    FOR EACH ROW
    BEGIN
        SELECT sequence_cities_prov.NEXTVAL INTO :NEW.id FROM dual;
    END;
/

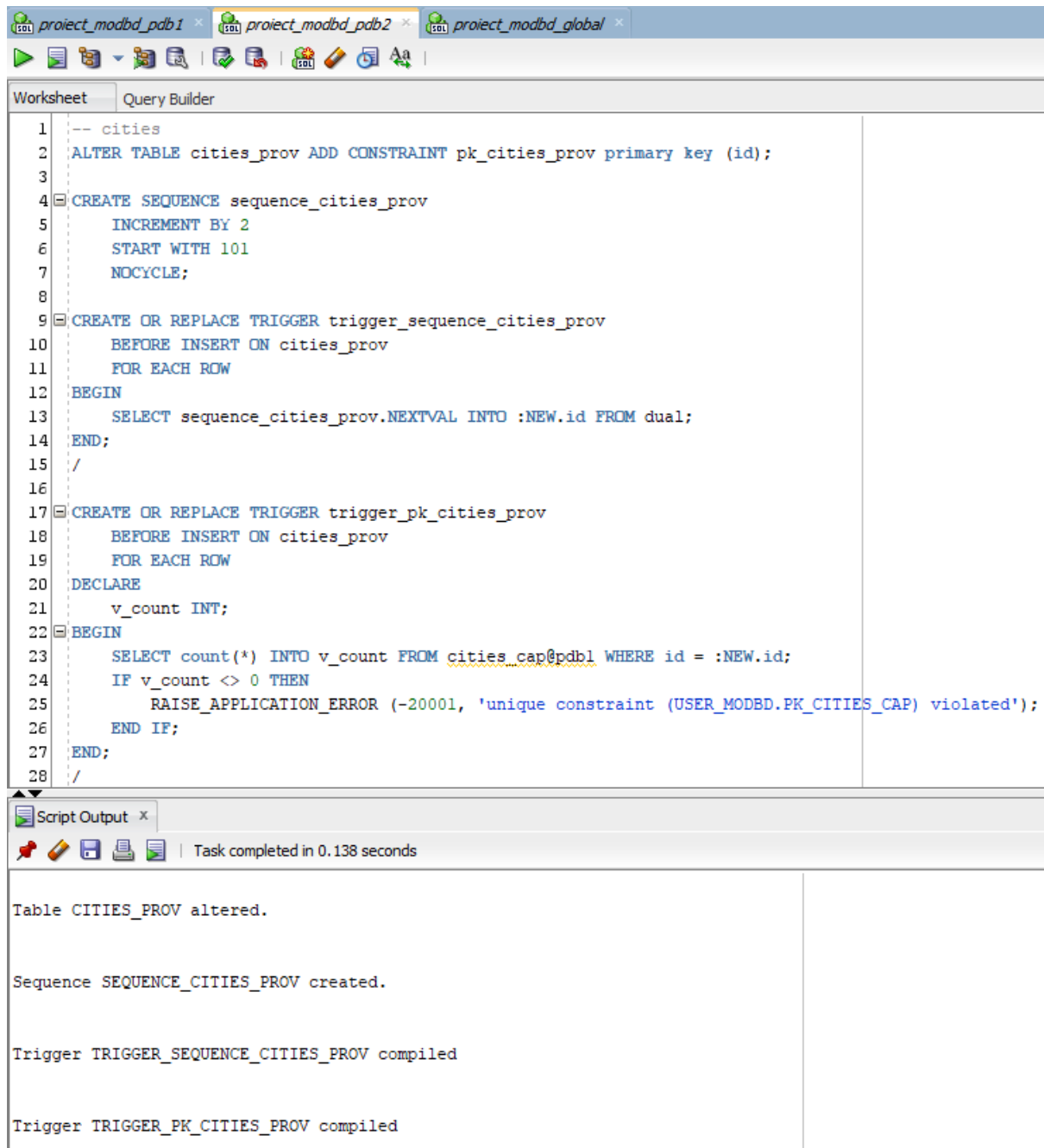
CREATE OR REPLACE TRIGGER trigger_pk_cities_prov
```



```

    BEFORE INSERT ON cities_prov
    FOR EACH ROW
DECLARE
    v_count INT;
BEGIN
    SELECT count(*) INTO v_count FROM cities_cap@pdb1 WHERE id = :NEW.id;
    IF v_count <> 0 THEN
        RAISE_APPLICATION_ERROR (-20001, 'unique constraint (USER_MODBD.PK_CITIES_CAP)
violated');
    END IF;
END;
/

```



The screenshot shows the Oracle SQL Developer interface. The top pane is the 'Query Builder' tab, displaying a SQL script for creating a primary key constraint and triggers for the 'cities\_prov' table. The script includes comments, an ALTER TABLE statement, a sequence creation, and two trigger definitions. The bottom pane is the 'Script Output' window, which shows the execution results of the script.

**Script Editor Content:**

```

1  -- cities
2  ALTER TABLE cities_prov ADD CONSTRAINT pk_cities_prov primary key (id);
3
4  CREATE SEQUENCE sequence_cities_prov
5      INCREMENT BY 2
6      START WITH 101
7      NOCYCLE;
8
9  CREATE OR REPLACE TRIGGER trigger_sequence_cities_prov
10     BEFORE INSERT ON cities_prov
11     FOR EACH ROW
12     BEGIN
13         SELECT sequence_cities_prov.NEXTVAL INTO :NEW.id FROM dual;
14     END;
15 /
16
17 CREATE OR REPLACE TRIGGER trigger_pk_cities_prov
18     BEFORE INSERT ON cities_prov
19     FOR EACH ROW
20     DECLARE
21         v_count INT;
22     BEGIN
23         SELECT count(*) INTO v_count FROM cities_cap@pdb1 WHERE id = :NEW.id;
24         IF v_count <> 0 THEN
25             RAISE_APPLICATION_ERROR (-20001, 'unique constraint (USER_MODBD.PK_CITIES_CAP) violated');
26         END IF;
27     END;
28 /

```

**Script Output Content:**

```

Table CITIES_PROV altered.

Sequence SEQUENCE_CITIES_PROV created.

Trigger TRIGGER_SEQUENCE_CITIES_PROV compiled

Trigger TRIGGER_PK_CITIES_PROV compiled

```

```

-- restaurants
ALTER TABLE restaurants_cap ADD CONSTRAINT pk_restaurants_cap PRIMARY KEY (id);

```

```
ALTER TABLE restaurants_cap ADD CONSTRAINT fk_restaurants_cap_cities_cap FOREIGN KEY
(city_id) REFERENCES cities_cap(id);
```

```
CREATE SEQUENCE sequence_restaurants_cap
  INCREMENT BY 2
  START WITH 100
  NOCYCLE;
```

```
CREATE OR REPLACE TRIGGER trigger_sequence_restaurants_cap
  BEFORE INSERT ON restaurants_cap
  FOR EACH ROW
BEGIN
  SELECT sequence_restaurants_cap.NEXTVAL INTO :NEW.id FROM dual;
END;
/
```

```
CREATE OR REPLACE TRIGGER trigger_pk_restaurants_cap
  BEFORE INSERT ON restaurants_cap
  FOR EACH ROW
DECLARE
  v_count INT;
BEGIN
  SELECT count(*) INTO v_count FROM restaurants_prov@pdb2 WHERE id = :NEW.id;
  IF v_count <> 0 THEN
    RAISE_APPLICATION_ERROR (-20001, 'unique constraint
(USER_MODBD.PK_RESTAURANTS_PROV) violated');
  END IF;
END;
/
```

The screenshot shows the Oracle SQL Developer interface. The top tab bar has three tabs: 'proiect\_modbd\_pdb1', 'proiect\_modbd\_pdb2', and 'proiect\_modbd\_global'. The 'Worksheet' tab is active, displaying a SQL script. The 'Script Output' tab is also open, showing the results of the script execution.

```

1  -- restaurants
2  ALTER TABLE restaurants_cap ADD CONSTRAINT pk_restaurants_cap PRIMARY KEY (id);
3  ALTER TABLE restaurants_cap ADD CONSTRAINT fk_restaurants_cap_cities_cap FOREIGN KEY (city_id) REFERENCES cities_cap(id);
4
5  CREATE SEQUENCE sequence_restaurants_cap
6      INCREMENT BY 2
7      START WITH 100
8      NOCYCLE;
9
10 CREATE OR REPLACE TRIGGER trigger_sequence_restaurants_cap
11     BEFORE INSERT ON restaurants_cap
12     FOR EACH ROW
13 BEGIN
14     SELECT sequence_restaurants_cap.NEXTVAL INTO :NEW.id FROM dual;
15 END;
16 /
17
18 CREATE OR REPLACE TRIGGER trigger_pk_restaurants_cap
19     BEFORE INSERT ON restaurants_cap
20     FOR EACH ROW
21 DECLARE
22     v_count INT;
23 BEGIN
24     SELECT count(*) INTO v_count FROM restaurants_prov@pdb2 WHERE id = :NEW.id;
25     IF v_count <> 0 THEN
26         RAISE_APPLICATION_ERROR (-20001, 'unique constraint (USER_MODBD.PK_RESTAURANTS_PROV) violated');
27     END IF;
28 END;
29 /

```

The Script Output tab shows the following messages:

```

Table RESTAURANTS_CAP altered.

Table RESTAURANTS_CAP altered.

Sequence SEQUENCE_RESTAURANTS_CAP created.

Trigger TRIGGER_SEQUENCE_RESTAURANTS_CAP compiled

Trigger TRIGGER_PK_RESTAURANTS_CAP compiled

```

```

-- restaurants
ALTER TABLE restaurants_prov ADD CONSTRAINT pk_restaurants_prov PRIMARY KEY (id);
ALTER TABLE restaurants_prov ADD CONSTRAINT fk_restaurants_prov_cities_prov FOREIGN KEY (city_id) REFERENCES cities_prov(id);

```

```

CREATE SEQUENCE sequence_restaurants_prov
    INCREMENT BY 2
    START WITH 101
    NOCYCLE;

```

```

CREATE OR REPLACE TRIGGER trigger_sequence_restaurants_prov
    BEFORE INSERT ON restaurants_prov
    FOR EACH ROW
BEGIN
    SELECT sequence_restaurants_prov.NEXTVAL INTO :NEW.id FROM dual;
END;
/

```

```

CREATE OR REPLACE TRIGGER trigger_pk_restaurants_prov
    BEFORE INSERT ON restaurants_prov
    FOR EACH ROW

```

```

DECLARE
  v_count INT;
BEGIN
  SELECT count(*) INTO v_count FROM restaurants_cap@pdb1 WHERE id = :NEW.id;
  IF v_count <> 0 THEN
    RAISE_APPLICATION_ERROR (-20001, 'unique constraint
(USER_MODBD.PK_RESTAURANTS_CAP) violated');
  END IF;
END;
/

```

The screenshot shows the SQL Developer interface with three tabs: 'project\_modbd\_pdb1', 'project\_modbd\_pdb2', and 'project\_modbd\_global'. The 'Query Builder' tab is active, displaying a SQL script. Below the script, the 'Script Output' window shows the results of the execution.

```

1  -- restaurants
2  ALTER TABLE restaurants_prov ADD CONSTRAINT pk_restaurants_prov PRIMARY KEY (id);
3  ALTER TABLE restaurants_prov ADD CONSTRAINT fk_restaurants_prov_cities_prov FOREIGN KEY (city_id) REFERENCES cities_prov(id);
4
5  CREATE SEQUENCE sequence_restaurants_prov
6      INCREMENT BY 2
7      START WITH 101
8      NOCYCLE;
9
10 CREATE OR REPLACE TRIGGER trigger_sequence_restaurants_prov
11     BEFORE INSERT ON restaurants_prov
12     FOR EACH ROW
13 BEGIN
14     SELECT sequence_restaurants_prov.NEXTVAL INTO :NEW.id FROM dual;
15 END;
16 /
17
18 CREATE OR REPLACE TRIGGER trigger_pk_restaurants_prov
19     BEFORE INSERT ON restaurants_prov
20     FOR EACH ROW
21 DECLARE
22     v_count INT;
23 BEGIN
24     SELECT count(*) INTO v_count FROM restaurants_cap@pdb1 WHERE id = :NEW.id;
25     IF v_count <> 0 THEN
26         RAISE_APPLICATION_ERROR (-20001, 'unique constraint (USER_MODBD.PK_RESTAURANTS_CAP) violated');
27     END IF;
28 END;
29 /

```

Script Output

Task completed in 0.05 seconds

```

Table RESTAURANTS_PROV altered.

Table RESTAURANTS_PROV altered.

Sequence SEQUENCE_RESTAURANTS_PROV created.

Trigger TRIGGER_SEQUENCE_RESTAURANTS_PROV compiled

Trigger TRIGGER_PK_RESTAURANTS_PROV compiled

```

```

-- types
ALTER TABLE types_cap ADD CONSTRAINT pk_types_cap primary key (id);

CREATE SEQUENCE sequence_types_cap
  INCREMENT BY 1
  START WITH 100
  NOCYCLE;

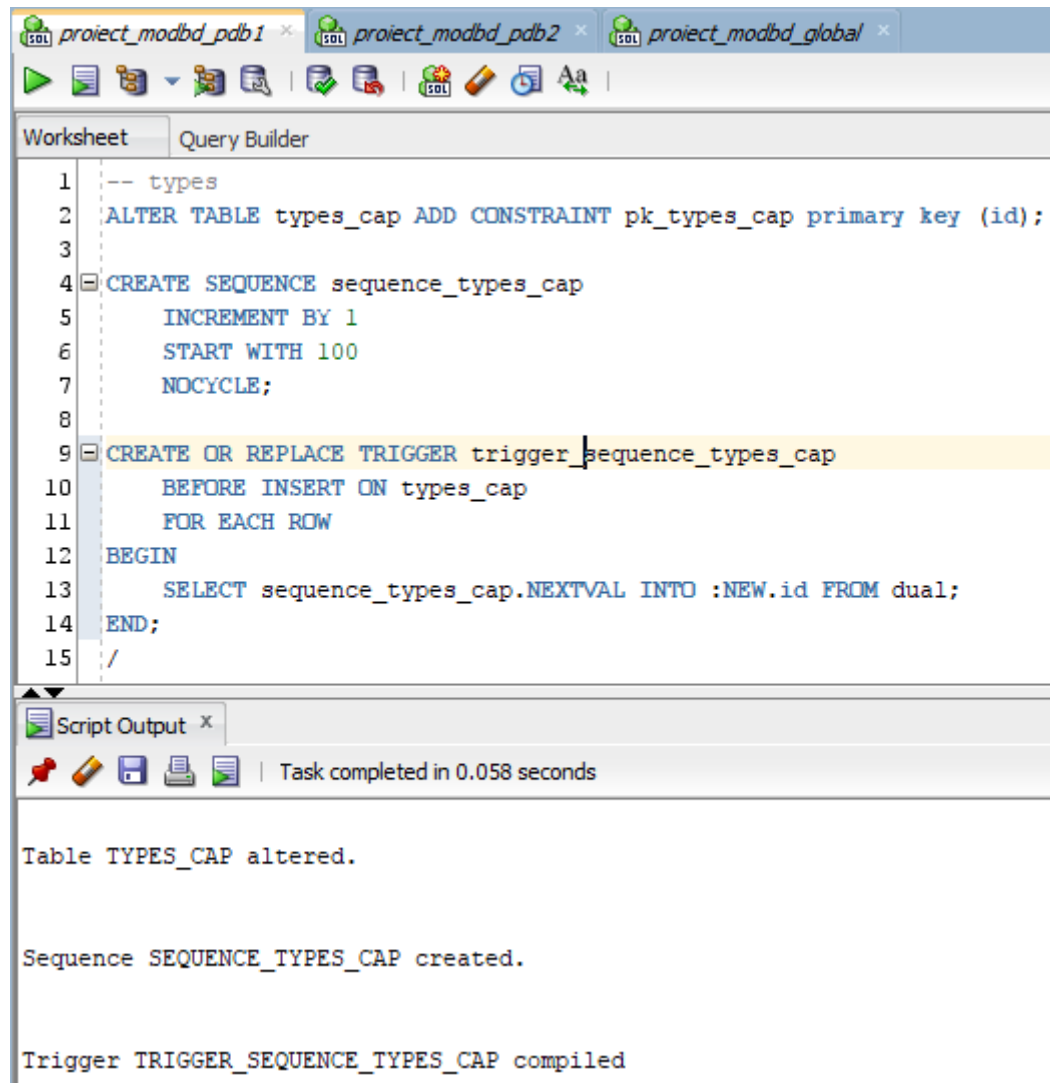
CREATE OR REPLACE TRIGGER trigger_sequence_types_cap
  BEFORE INSERT ON types_cap
  FOR EACH ROW

```

```

BEGIN
    SELECT sequence_types_cap.NEXTVAL INTO :NEW.id FROM dual;
END;
/

```



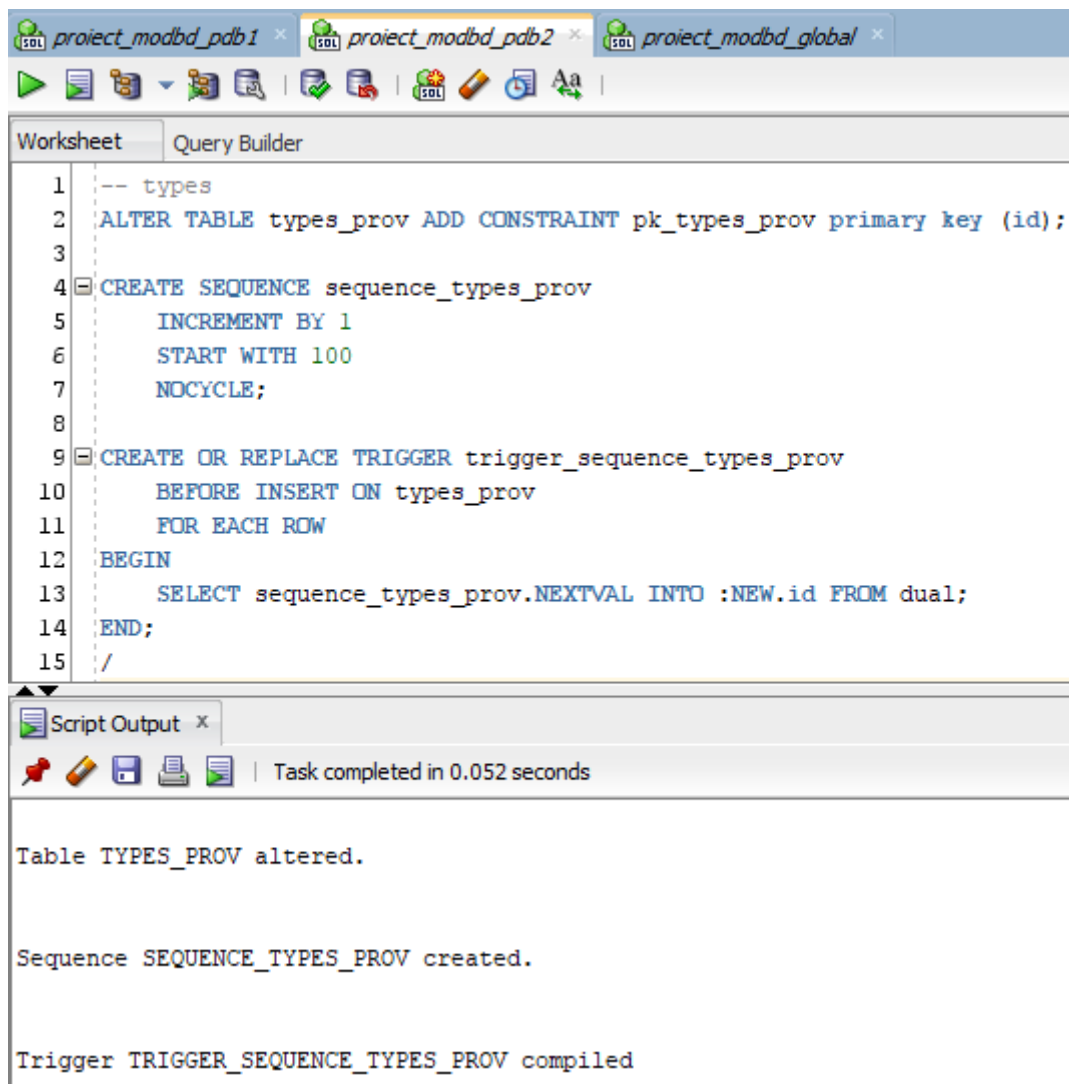
```

-- types
ALTER TABLE types_prov ADD CONSTRAINT pk_types_prov primary key (id);

CREATE SEQUENCE sequence_types_prov
    INCREMENT BY 1
    START WITH 100
    NOCYCLE;

CREATE OR REPLACE TRIGGER trigger_sequence_types_prov
    BEFORE INSERT ON types_prov
    FOR EACH ROW
BEGIN
    SELECT sequence_types_prov.NEXTVAL INTO :NEW.id FROM dual;
END;
/

```



```

-- employees
ALTER TABLE employees_cap ADD CONSTRAINT pk_employees_cap PRIMARY KEY (id);
ALTER TABLE employees_cap ADD CONSTRAINT fk_employees_cap_restaurants_cap FOREIGN
KEY (restaurant_id) REFERENCES restaurants_cap(id);
ALTER TABLE employees_cap ADD CONSTRAINT fk_employees_cap_type_cap FOREIGN KEY
(type_id) REFERENCES types_cap(id);
  
```

```

CREATE SEQUENCE sequence_employees_cap
  INCREMENT BY 2
  START WITH 100
  NOCYCLE;
  
```

```

CREATE OR REPLACE TRIGGER trigger_sequence_employees_cap
  BEFORE INSERT ON employees_cap
  FOR EACH ROW
BEGIN
  SELECT sequence_employees_cap.NEXTVAL INTO :NEW.id FROM dual;
END;
/
  
```

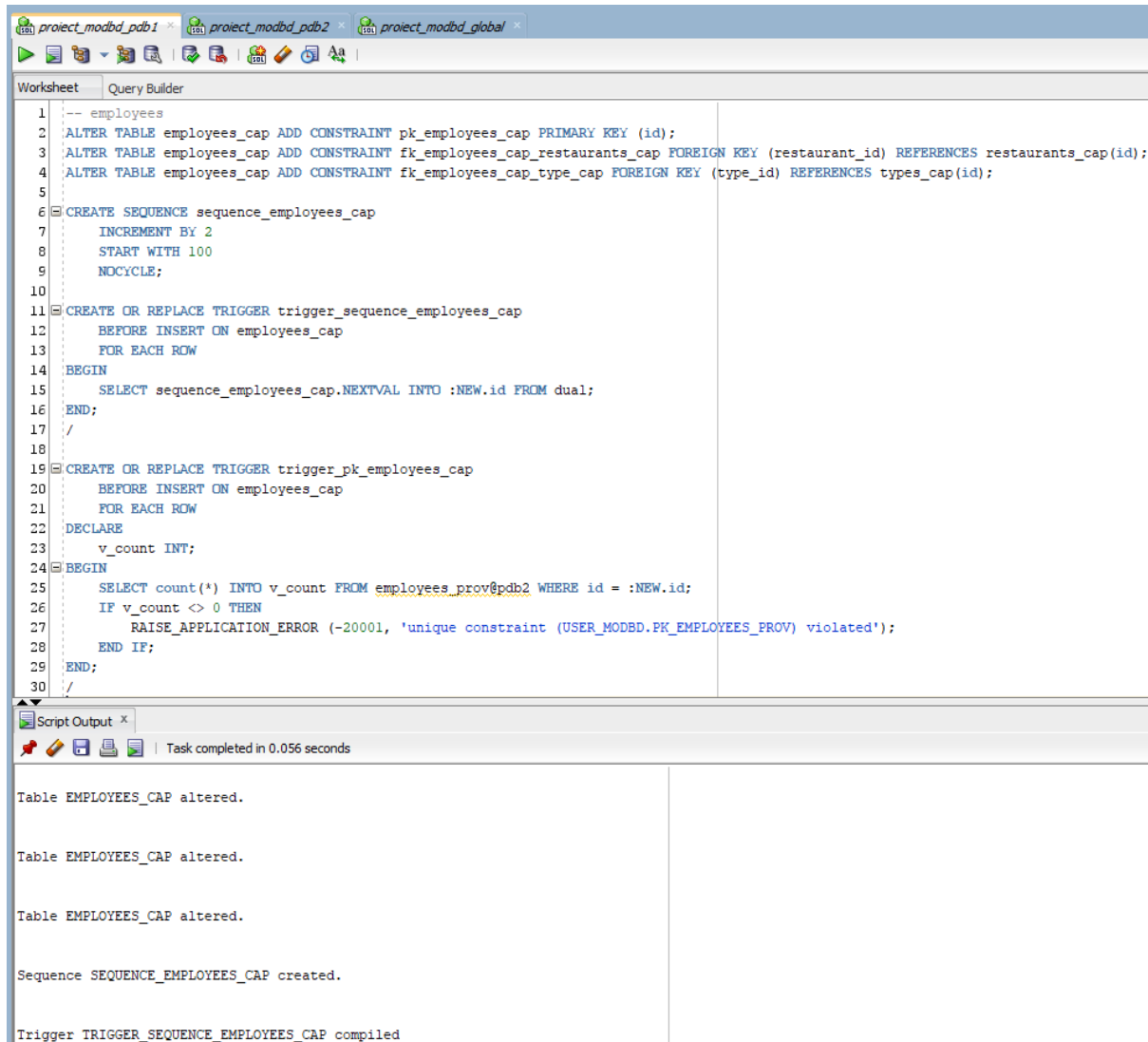
```

CREATE OR REPLACE TRIGGER trigger_pk_employees_cap
  BEFORE INSERT ON employees_cap
  FOR EACH ROW
DECLARE
  
```

```

v_count INT;
BEGIN
  SELECT count(*) INTO v_count FROM employees_prov@pdb2 WHERE id = :NEW.id;
  IF v_count <> 0 THEN
    RAISE_APPLICATION_ERROR (-20001, 'unique constraint
(USER_MODBD.PK_EMPLOYEES_PROV) violated');
  END IF;
END;
/

```



```

-- employees
ALTER TABLE employees_prov ADD CONSTRAINT pk_employees_prov PRIMARY KEY (id);
ALTER TABLE employees_prov ADD CONSTRAINT fk_employees_prov_restaurants_prov FOREIGN
KEY (restaurant_id) REFERENCES restaurants_prov(id);
ALTER TABLE employees_prov ADD CONSTRAINT fk_employees_prov_types_prov FOREIGN KEY
(type_id) REFERENCES types_prov(id);

CREATE SEQUENCE sequence_employees_prov
  INCREMENT BY 2
  START WITH 101
  NOCYCLE;

CREATE OR REPLACE TRIGGER trigger_sequence_employees_prov
  BEFORE INSERT ON employees_prov

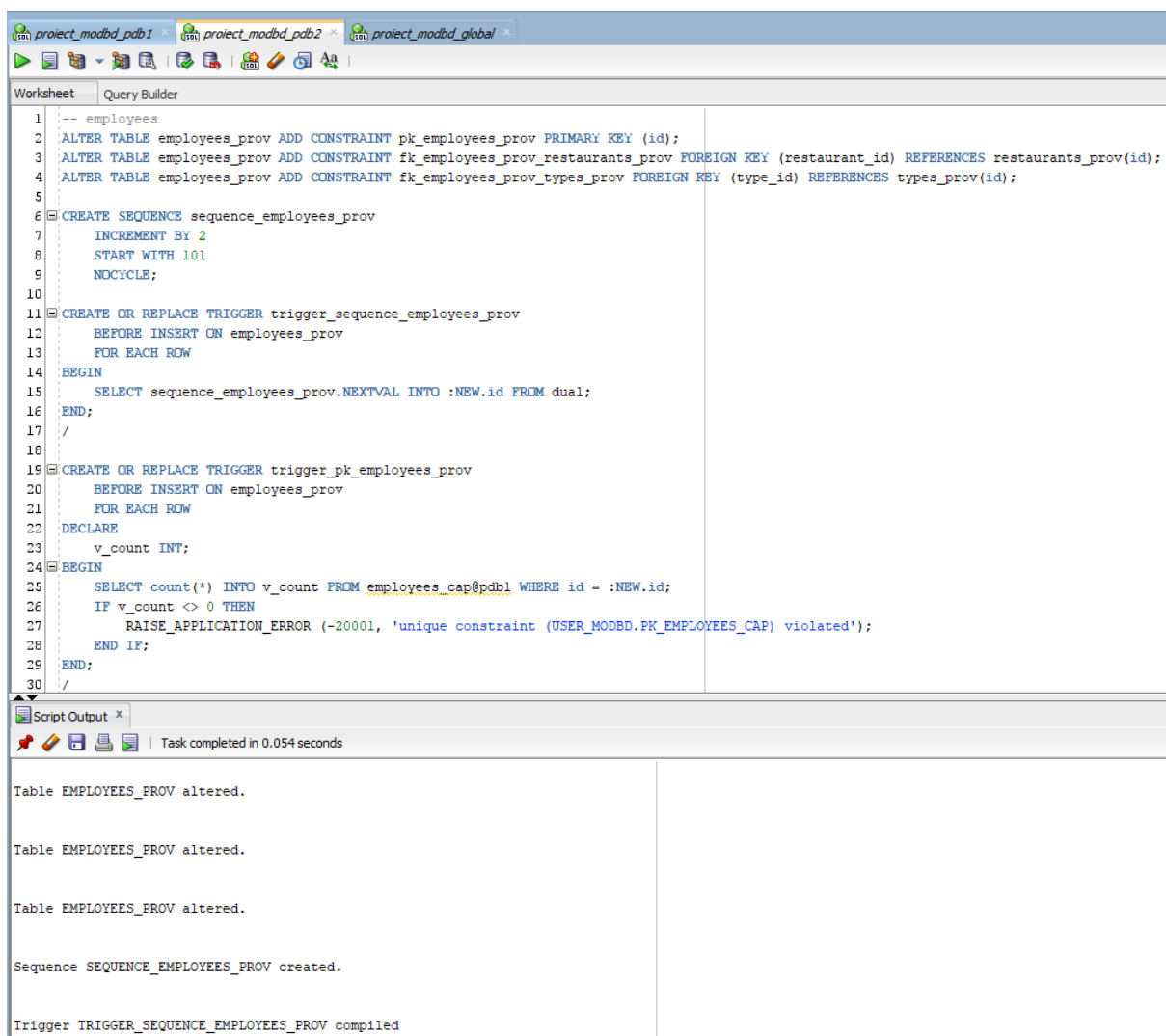
```

```

    FOR EACH ROW
BEGIN
    SELECT sequence_employees_prov.NEXTVAL INTO :NEW.id FROM dual;
END;
/

CREATE OR REPLACE TRIGGER trigger_pk_employees_prov
    BEFORE INSERT ON employees_prov
    FOR EACH ROW
DECLARE
    v_count INT;
BEGIN
    SELECT count(*) INTO v_count FROM employees_cap@pdb1 WHERE id = :NEW.id;
    IF v_count <> 0 THEN
        RAISE_APPLICATION_ERROR (-20001, 'unique constraint
(USER_MODBD.PK_EMPLOYEES_CAP) violated');
    END IF;
END;
/

```



The screenshot shows the Oracle SQL Developer interface. The top pane displays a SQL script with the following content:

```

1  -- employees
2  ALTER TABLE employees_prov ADD CONSTRAINT pk_employees_prov PRIMARY KEY (id);
3  ALTER TABLE employees_prov ADD CONSTRAINT fk_employees_prov_restaurants_prov FOREIGN KEY (restaurant_id) REFERENCES restaurants_prov(id);
4  ALTER TABLE employees_prov ADD CONSTRAINT fk_employees_prov_types_prov FOREIGN KEY (type_id) REFERENCES types_prov(id);
5
6  CREATE SEQUENCE sequence_employees_prov
7      INCREMENT BY 2
8      START WITH 101
9      NOCYCLE;
10
11 CREATE OR REPLACE TRIGGER trigger_sequence_employees_prov
12     BEFORE INSERT ON employees_prov
13     FOR EACH ROW
14 BEGIN
15     SELECT sequence_employees_prov.NEXTVAL INTO :NEW.id FROM dual;
16 END;
17 /
18
19 CREATE OR REPLACE TRIGGER trigger_pk_employees_prov
20     BEFORE INSERT ON employees_prov
21     FOR EACH ROW
22 DECLARE
23     v_count INT;
24 BEGIN
25     SELECT count(*) INTO v_count FROM employees_cap@pdb1 WHERE id = :NEW.id;
26     IF v_count <> 0 THEN
27         RAISE_APPLICATION_ERROR (-20001, 'unique constraint (USER_MODBD.PK_EMPLOYEES_CAP) violated');
28     END IF;
29 END;
30 /

```

The bottom pane shows the 'Script Output' window with the following messages:

```

Table EMPLOYEES_PROV altered.

Table EMPLOYEES_PROV altered.

Table EMPLOYEES_PROV altered.

Sequence SEQUENCE_EMPLOYEES_PROV created.

Trigger TRIGGER_SEQUENCE_EMPLOYEES_PROV compiled

```

At the top of the script output window, it says 'Task completed in 0.054 seconds'.

```

-- menus
ALTER TABLE menus_cap ADD CONSTRAINT pk_menus_cap PRIMARY KEY (id);

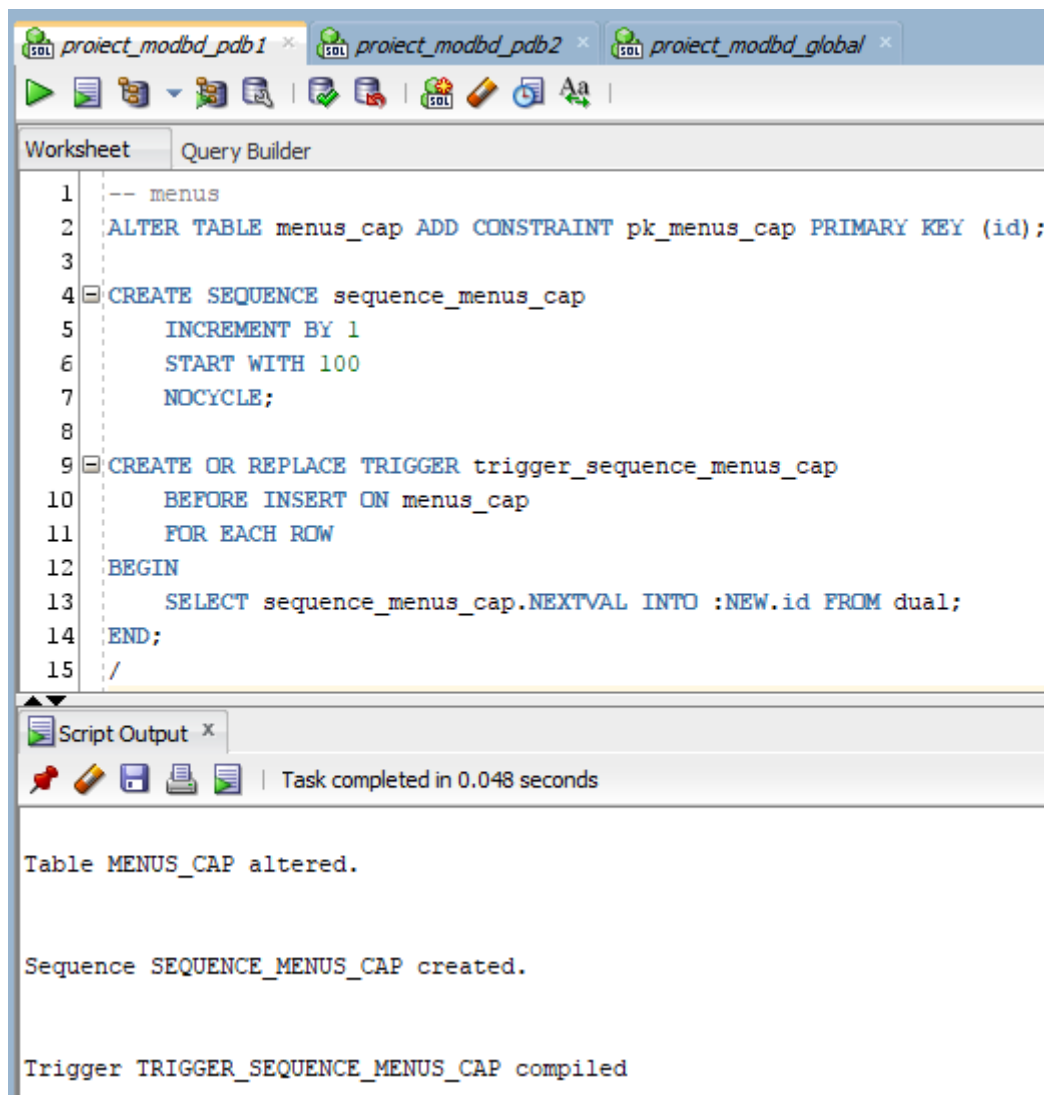
CREATE SEQUENCE sequence_menus_cap
    INCREMENT BY 1

```

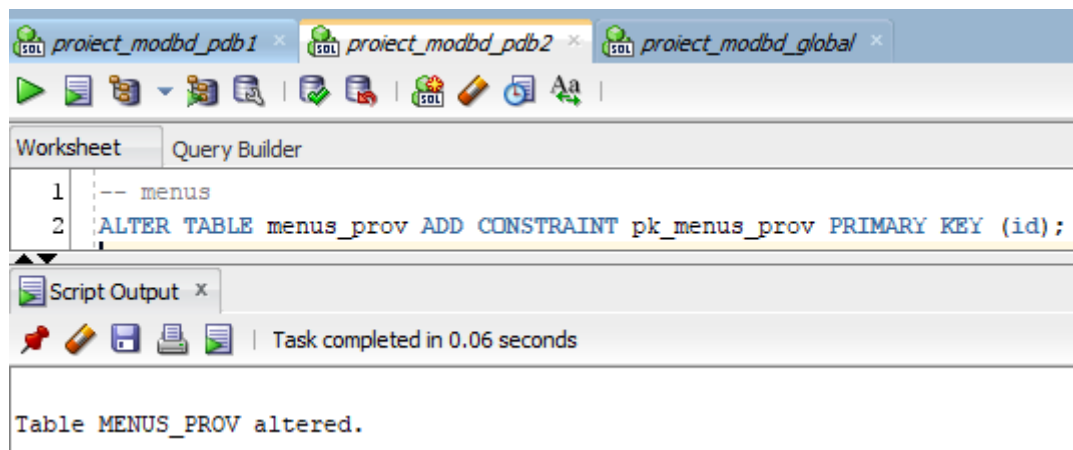


```
START WITH 100  
NOCYCLE;
```

```
CREATE OR REPLACE TRIGGER trigger_sequence_menus_cap  
  BEFORE INSERT ON menus_cap  
  FOR EACH ROW  
BEGIN  
  SELECT sequence_menus_cap.NEXTVAL INTO :NEW.id FROM dual;  
END;  
/
```



```
-- menus  
ALTER TABLE menus_prov ADD CONSTRAINT pk_menus_prov PRIMARY KEY (id);
```



```
-- drinks
ALTER TABLE drinks_cap ADD CONSTRAINT pk_drinks_cap PRIMARY KEY (id);
ALTER TABLE drinks_cap ADD CONSTRAINT fk_drinks_cap_menus_cap FOREIGN KEY (menu_id)
REFERENCES menus_cap(id);
```

```
CREATE SEQUENCE sequence_drinks_cap
INCREMENT BY 1
START WITH 100
NOCYCLE;
```

```
CREATE OR REPLACE TRIGGER trigger_sequence_drinks_cap
BEFORE INSERT ON drinks_cap
FOR EACH ROW
BEGIN
  SELECT sequence_drinks_cap.NEXTVAL INTO :NEW.id FROM dual;
END;
/
```

The screenshot shows the SQL Developer interface with three tabs: `proiect_modbd_pdb1`, `proiect_modbd_pdb2`, and `proiect_modbd_global`. The `Worksheet` tab is active, displaying a SQL script. The `Script Output` tab shows the execution results.

```

1  -- drinks
2  ALTER TABLE drinks_cap ADD CONSTRAINT pk_drinks_cap PRIMARY KEY (id);
3  ALTER TABLE drinks_cap ADD CONSTRAINT fk_drinks_cap_menus_cap FOREIGN KEY (menu_id) REFERENCES menus_cap(id);
4
5  CREATE SEQUENCE sequence_drinks_cap
6      INCREMENT BY 1
7      START WITH 100
8      NOCYCLE;
9
10 CREATE OR REPLACE TRIGGER trigger_sequence_drinks_cap
11     BEFORE INSERT ON drinks_cap
12     FOR EACH ROW
13 BEGIN
14     SELECT sequence_drinks_cap.NEXTVAL INTO :NEW.id FROM dual;
15 END;
16 /

```

Script Output:

```

Table DRINKS_CAP altered.

Table DRINKS_CAP altered.

Sequence SEQUENCE_DRINKS_CAP created.

Trigger TRIGGER_SEQUENCE_DRINKS_CAP compiled

```

```

-- drinks
ALTER TABLE drinks_prov ADD CONSTRAINT pk_drinks_prov PRIMARY KEY (id);
ALTER TABLE drinks_prov ADD CONSTRAINT fk_drinks_prov_menus_prov FOREIGN KEY
(menu_id) REFERENCES menus_prov(id);

```

The screenshot shows the SQL Developer interface with three tabs: `proiect_modbd_pdb1`, `proiect_modbd_pdb2`, and `proiect_modbd_global`. The `Worksheet` tab is active, displaying a SQL script. The `Script Output` tab shows the execution results.

```

1  -- drinks
2  ALTER TABLE drinks_prov ADD CONSTRAINT pk_drinks_prov PRIMARY KEY (id);
3  ALTER TABLE drinks_prov ADD CONSTRAINT fk_drinks_prov_menus_prov FOREIGN KEY (menu_id) REFERENCES menus_prov(id);

```

Script Output:

```

Table DRINKS_PROV altered.

Table DRINKS_PROV altered.

```

```

-- dishes
ALTER TABLE dishes_cap ADD CONSTRAINT pk_dishes_cap PRIMARY KEY (id);
ALTER TABLE dishes_cap ADD CONSTRAINT fk_dishes_cap_menus_cap FOREIGN KEY
(menu_id) REFERENCES menus_cap(id);

```

```

CREATE SEQUENCE sequence_dishes_cap
    INCREMENT BY 1
    START WITH 100
    NOCYCLE;

```

```

CREATE OR REPLACE TRIGGER trigger_sequence_dishes_cap

```

```

BEFORE INSERT ON dishes_cap
FOR EACH ROW
BEGIN
    SELECT sequence_dishes_cap.NEXTVAL INTO :NEW.id FROM dual;
END;
/

```

The screenshot shows the SQL Developer interface with three tabs: project\_modbd\_pdb1, project\_modbd\_pdb2, and project\_modbd\_global. The 'Query Builder' tab is active, displaying a script for the dishes\_cap table. The script includes altering the table to add primary and foreign key constraints, creating a sequence, and creating a trigger. The 'Script Output' pane below shows the execution results.

```

1  -- dishes
2  ALTER TABLE dishes_cap ADD CONSTRAINT pk_dishes_cap PRIMARY KEY (id);
3  ALTER TABLE dishes_cap ADD CONSTRAINT fk_dishes_cap_menus_cap FOREIGN KEY (menu_id) REFERENCES menus_cap(id);
4
5  CREATE SEQUENCE sequence_dishes_cap
6      INCREMENT BY 1
7      START WITH 100
8      NOCYCLE;
9
10 CREATE OR REPLACE TRIGGER trigger_sequence_dishes_cap
11     BEFORE INSERT ON dishes_cap
12     FOR EACH ROW
13 BEGIN
14     SELECT sequence_dishes_cap.NEXTVAL INTO :NEW.id FROM dual;
15 END;
16 /

```

Script Output x

Task completed in 0.052 seconds

```

Table DISHES_CAP altered.

Table DISHES_CAP altered.

Sequence SEQUENCE_DISHES_CAP created.

Trigger TRIGGER_SEQUENCE_DISHES_CAP compiled

```

```

-- dishes
ALTER TABLE dishes_prov ADD CONSTRAINT pk_dishes_prov PRIMARY KEY (id);
ALTER TABLE dishes_prov ADD CONSTRAINT fk_dishes_prov_menus_prov FOREIGN KEY
(menu_id) REFERENCES menus_prov(id);

```

The screenshot shows the SQL Developer interface with three tabs: project\_modbd\_pdb1, project\_modbd\_pdb2, and project\_modbd\_global. The 'Query Builder' tab is active, displaying a script for the dishes\_prov table. The script includes altering the table to add primary and foreign key constraints. The 'Script Output' pane below shows the execution results.

```

1  -- dishes
2  ALTER TABLE dishes_prov ADD CONSTRAINT pk_dishes_prov PRIMARY KEY (id);
3  ALTER TABLE dishes_prov ADD CONSTRAINT fk_dishes_prov_menus_prov FOREIGN KEY (menu_id) REFERENCES menus_prov(id);

```

Script Output x

Task completed in 0.043 seconds

```

Table DISHES_PROV altered.

Table DISHES_PROV altered.

```

```

-- orders
ALTER TABLE orders_cap ADD CONSTRAINT pk_orders_cap PRIMARY KEY (id);
ALTER TABLE orders_cap ADD CONSTRAINT fk_orders_cap_employees_cap FOREIGN KEY
(waiter_id) REFERENCES employees_cap(id);

```

```

CREATE SEQUENCE sequence_orders_cap
  INCREMENT BY 2
  START WITH 100
  NOCYCLE;

CREATE OR REPLACE TRIGGER trigger_sequence_orders_cap
  BEFORE INSERT ON orders_cap
  FOR EACH ROW
BEGIN
  SELECT sequence_orders_cap.NEXTVAL INTO :NEW.id FROM dual;
END;
/

CREATE OR REPLACE TRIGGER trigger_pk_orders_cap
  BEFORE INSERT ON orders_cap
  FOR EACH ROW
DECLARE
  v_count INT;
BEGIN
  SELECT count(*) INTO v_count FROM orders_prov@pdb2 WHERE id = :NEW.id;
  IF v_count <> 0 THEN
    RAISE_APPLICATION_ERROR (-20001, 'unique constraint
(USER_MODBD.PK_ORDERS_PROV) violated');
  END IF;
END;
/

```

The screenshot shows the Oracle SQL Developer interface. The top pane is the 'Script Editor' with a tab labeled 'project\_modbd\_pdb1'. It contains the following SQL script:

```

1  -- orders
2  ALTER TABLE orders_cap ADD CONSTRAINT pk_orders_cap PRIMARY KEY (id);
3  ALTER TABLE orders_cap ADD CONSTRAINT fk_orders_cap_employees_cap FOREIGN KEY (waiter_id) REFERENCES employees_cap(id);
4
5  CREATE SEQUENCE sequence_orders_cap
6      INCREMENT BY 2
7      START WITH 100
8      NOCYCLE;
9
10 CREATE OR REPLACE TRIGGER trigger_sequence_orders_cap
11     BEFORE INSERT ON orders_cap
12     FOR EACH ROW
13 BEGIN
14     SELECT sequence_orders_cap.NEXTVAL INTO :NEW.id FROM dual;
15 END;
16 /
17
18 CREATE OR REPLACE TRIGGER trigger_pk_orders_cap
19     BEFORE INSERT ON orders_cap
20     FOR EACH ROW
21 DECLARE
22     v_count INT;
23 BEGIN
24     SELECT count(*) INTO v_count FROM orders_prov@pdb2 WHERE id = :NEW.id;
25     IF v_count <> 0 THEN
26         RAISE_APPLICATION_ERROR (-20001, 'unique constraint (USER_MODBD.PK_ORDERS_PROV) violated');
27     END IF;
28 END;
29 /

```

The bottom pane is the 'Script Output' window, showing the execution results:

```

Table ORDERS_CAP altered.

Table ORDERS_CAP altered.

Sequence SEQUENCE_ORDERS_CAP created.

Trigger TRIGGER_SEQUENCE_ORDERS_CAP compiled

Trigger TRIGGER_PK_ORDERS_CAP compiled

```

```

-- orders
ALTER TABLE orders_prov ADD CONSTRAINT pk_orders_prov PRIMARY KEY (id);
ALTER TABLE orders_prov ADD CONSTRAINT fk_orders_prov_employees_prov FOREIGN KEY (waiter_id) REFERENCES employees_prov(id);

CREATE SEQUENCE sequence_orders_prov
    INCREMENT BY 2
    START WITH 101
    NOCYCLE;

CREATE OR REPLACE TRIGGER trigger_sequence_orders_prov
    BEFORE INSERT ON orders_prov
    FOR EACH ROW
BEGIN
    SELECT sequence_orders_prov.NEXTVAL INTO :NEW.id FROM dual;
END;
/

CREATE OR REPLACE TRIGGER trigger_pk_orders_prov
    BEFORE INSERT ON orders_prov
    FOR EACH ROW

```

```

DECLARE
  v_count INT;
BEGIN
  SELECT count(*) INTO v_count FROM orders_cap@pdb1 WHERE id = :NEW.id;
  IF v_count <> 0 THEN
    RAISE_APPLICATION_ERROR (-20001, 'unique constraint
(USER_MODBD.PK_ORDERS_CAP) violated');
  END IF;
END;
/

```

The screenshot shows the Oracle SQL Developer interface. The top pane displays a SQL script with the following content:

```

1  -- orders
2  ALTER TABLE orders_prov ADD CONSTRAINT pk_orders_prov PRIMARY KEY (id);
3  ALTER TABLE orders_prov ADD CONSTRAINT fk_orders_prov_employees_prov FOREIGN KEY (waiter_id) REFERENCES employees_prov(id);
4
5  CREATE SEQUENCE sequence_orders_prov
6    INCREMENT BY 2
7    START WITH 101
8    NOCYCLE;
9
10 CREATE OR REPLACE TRIGGER trigger_sequence_orders_prov
11   BEFORE INSERT ON orders_prov
12   FOR EACH ROW
13 BEGIN
14   SELECT sequence_orders_prov.NEXTVAL INTO :NEW.id FROM dual;
15 END;
16 /
17
18 CREATE OR REPLACE TRIGGER trigger_pk_orders_prov
19   BEFORE INSERT ON orders_prov
20   FOR EACH ROW
21 DECLARE
22   v_count INT;
23 BEGIN
24   SELECT count(*) INTO v_count FROM orders_cap@pdb1 WHERE id = :NEW.id;
25   IF v_count <> 0 THEN
26     RAISE_APPLICATION_ERROR (-20001, 'unique constraint (USER_MODBD.PK_ORDERS_CAP) violated');
27   END IF;
28 END;
29 /

```

The bottom pane shows the 'Script Output' tab with the following execution results:

```

Table ORDERS_PROV altered.

Table ORDERS_PROV altered.

Sequence SEQUENCE_ORDERS_PROV created.

Trigger TRIGGER_SEQUENCE_ORDERS_PROV compiled

Trigger TRIGGER_PK_ORDERS_PROV compiled

```

```

-- orders_drinks
ALTER TABLE orders_drinks_cap ADD CONSTRAINT pk_orders_drinks_cap PRIMARY KEY (id);
ALTER TABLE orders_drinks_cap ADD CONSTRAINT fk_orders_drinks_cap_orders_cap FOREIGN
KEY (order_id) REFERENCES orders_cap(id);
ALTER TABLE orders_drinks_cap ADD CONSTRAINT fk_orders_drinks_cap_drinks_cap FOREIGN
KEY (drink_id) REFERENCES drinks_cap(id);

CREATE SEQUENCE sequence_orders_drinks_cap
  INCREMENT BY 2
  START WITH 100
  NOCYCLE;

```

```

CREATE OR REPLACE TRIGGER trigger_sequence_orders_drinks_cap
  BEFORE INSERT ON orders_drinks_cap
  FOR EACH ROW
BEGIN
  SELECT sequence_orders_drinks_cap.NEXTVAL INTO :NEW.id FROM dual;
END;
/

```

```

CREATE OR REPLACE TRIGGER trigger_pk_orders_drinks_cap
  BEFORE INSERT ON orders_drinks_cap
  FOR EACH ROW
DECLARE
  v_count INT;
BEGIN
  SELECT count(*) INTO v_count FROM orders_drinks_prov@pdb2 WHERE id = :NEW.id;
  IF v_count <> 0 THEN
    RAISE_APPLICATION_ERROR (-20001, 'unique constraint
(USER_MODBD.PK_ORDERS_DRINKS_PROV) violated');
  END IF;
END;
/

```



The screenshot shows the Oracle SQL Developer interface. The top pane displays a SQL script in the Query Builder, and the bottom pane shows the Script Output window with the results of the script execution.

**SQL Script (Query Builder):**

```

1  -- orders_drinks
2  ALTER TABLE orders_drinks_cap ADD CONSTRAINT pk_orders_drinks_cap PRIMARY KEY (id);
3  ALTER TABLE orders_drinks_cap ADD CONSTRAINT fk_orders_drinks_cap_orders_cap FOREIGN KEY (order_id) REFERENCES orders_cap(id);
4  ALTER TABLE orders_drinks_cap ADD CONSTRAINT fk_orders_drinks_cap_drinks_cap FOREIGN KEY (drink_id) REFERENCES drinks_cap(id);
5
6  CREATE SEQUENCE sequence_orders_drinks_cap
7      INCREMENT BY 2
8      START WITH 100
9      NOCYCLE;
10
11 CREATE OR REPLACE TRIGGER trigger_sequence_orders_drinks_cap
12     BEFORE INSERT ON orders_drinks_cap
13     FOR EACH ROW
14 BEGIN
15     SELECT sequence_orders_drinks_cap.NEXTVAL INTO :NEW.id FROM dual;
16 END;
17 /
18
19 CREATE OR REPLACE TRIGGER trigger_pk_orders_drinks_cap
20     BEFORE INSERT ON orders_drinks_cap
21     FOR EACH ROW
22 DECLARE
23     v_count INT;
24 BEGIN
25     SELECT count(*) INTO v_count FROM orders_drinks_prov@pdb2 WHERE id = :NEW.id;
26     IF v_count <> 0 THEN
27         RAISE_APPLICATION_ERROR (-20001, 'unique constraint (USER_MOBBD.PK_ORDERS_DRINKS_PROV) violated');
28     END IF;
29 END;

```

**Script Output:**

```

Table ORDERS_DRINKS_CAP altered.

Table ORDERS_DRINKS_CAP altered.

Table ORDERS_DRINKS_CAP altered.

Sequence SEQUENCE_ORDERS_DRINKS_CAP created.

Trigger TRIGGER_SEQUENCE_ORDERS_DRINKS_CAP compiled

Trigger TRIGGER_PK_ORDERS_DRINKS_CAP compiled

```

```

-- orders_drinks
ALTER TABLE orders_drinks_prov ADD CONSTRAINT pk_orders_drinks_prov PRIMARY KEY (id);
ALTER TABLE orders_drinks_prov ADD CONSTRAINT fk_orders_drinks_prov_orders_prov
FOREIGN KEY (order_id) REFERENCES orders_prov(id);
ALTER TABLE orders_drinks_prov ADD CONSTRAINT fk_orders_drinks_prov_drinks_prov
FOREIGN KEY (drink_id) REFERENCES drinks_prov(id);

```

```

CREATE SEQUENCE sequence_orders_drinks_prov
    INCREMENT BY 2
    START WITH 101
    NOCYCLE;

```

```

CREATE OR REPLACE TRIGGER trigger_sequence_orders_drinks_prov
    BEFORE INSERT ON orders_drinks_prov
    FOR EACH ROW
BEGIN
    SELECT sequence_orders_drinks_prov.NEXTVAL INTO :NEW.id FROM dual;
END;
/

```

```

CREATE OR REPLACE TRIGGER trigger_pk_orders_drinks_prov

```

```

    BEFORE INSERT ON orders_drinks_prov
    FOR EACH ROW
DECLARE
    v_count INT;
BEGIN
    SELECT count(*) INTO v_count FROM orders_drinks_cap@pdb1 WHERE id = :NEW.id;
    IF v_count <> 0 THEN
        RAISE_APPLICATION_ERROR (-20001, 'unique constraint
(USER_MODBD.PK_ORDERS_DRINKS_CAP) violated');
    END IF;
END;
/

```

The screenshot shows the SQL Developer interface. The top pane displays a SQL script with the following content:

```

1  -- orders_drinks
2  ALTER TABLE orders_drinks_prov ADD CONSTRAINT pk_orders_drinks_prov PRIMARY KEY (id);
3  ALTER TABLE orders_drinks_prov ADD CONSTRAINT fk_orders_drinks_prov_orders_prov FOREIGN KEY (order_id) REFERENCES orders_prov(id);
4  ALTER TABLE orders_drinks_prov ADD CONSTRAINT fk_orders_drinks_prov_drinks_prov FOREIGN KEY (drink_id) REFERENCES drinks_prov(id);
5
6  CREATE SEQUENCE sequence_orders_drinks_prov
7      INCREMENT BY 2
8      START WITH 101
9      NOCYCLE;
10
11 CREATE OR REPLACE TRIGGER trigger_sequence_orders_drinks_prov
12     BEFORE INSERT ON orders_drinks_prov
13     FOR EACH ROW
14 BEGIN
15     SELECT sequence_orders_drinks_prov.NEXTVAL INTO :NEW.id FROM dual;
16 END;
17 /
18
19 CREATE OR REPLACE TRIGGER trigger_pk_orders_drinks_prov
20     BEFORE INSERT ON orders_drinks_prov
21     FOR EACH ROW
22 DECLARE
23     v_count INT;
24 BEGIN
25     SELECT count(*) INTO v_count FROM orders_drinks_cap@pdb1 WHERE id = :NEW.id;
26     IF v_count <> 0 THEN
27         RAISE_APPLICATION_ERROR (-20001, 'unique constraint (USER_MODBD.PK_ORDERS_DRINKS_CAP) violated');
28     END IF;
29 END;

```

The bottom pane shows the 'Script Output' window with the following messages:

```

Table ORDERS_DRINKS_PROV altered.

Table ORDERS_DRINKS_PROV altered.

Table ORDERS_DRINKS_PROV altered.

Sequence SEQUENCE_ORDERS_DRINKS_PROV created.

Trigger TRIGGER_SEQUENCE_ORDERS_DRINKS_PROV compiled

Trigger TRIGGER_PK_ORDERS_DRINKS_PROV compiled

```

```

-- chefs_orders_dishes
ALTER TABLE chefs_orders_dishes_cap ADD CONSTRAINT pk_chefs_orders_dishes_cap
PRIMARY KEY (id);
ALTER TABLE chefs_orders_dishes_cap ADD CONSTRAINT
fk_chefs_orders_dishes_cap_orders_cap FOREIGN KEY (order_id) REFERENCES orders_cap(id);
ALTER TABLE chefs_orders_dishes_cap ADD CONSTRAINT
fk_chefs_orders_dishes_cap_dishes_cap FOREIGN KEY (dish_id) REFERENCES dishes_cap(id);

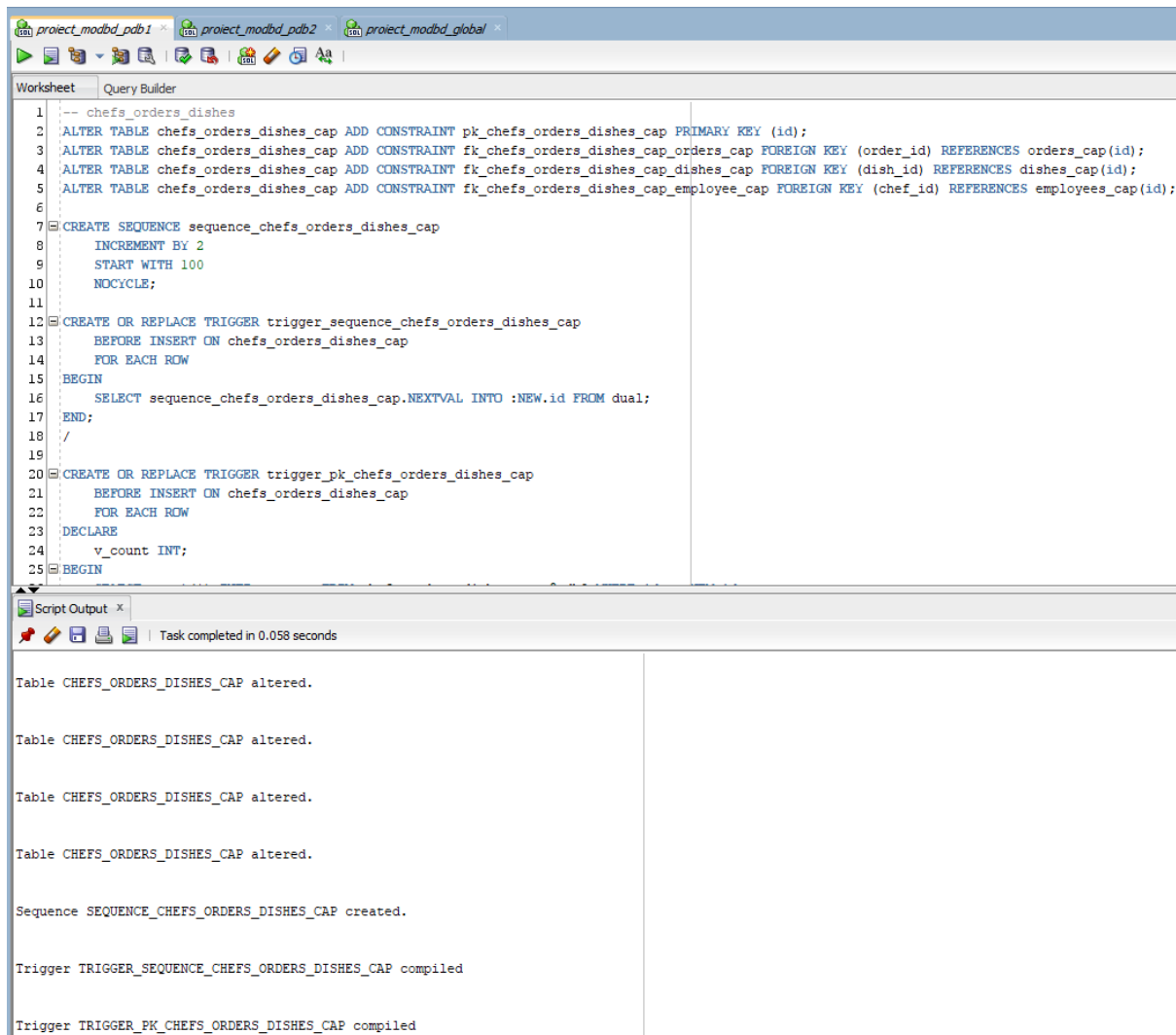
```

```
ALTER TABLE chefs_orders_dishes_cap ADD CONSTRAINT  
fk_chefs_orders_dishes_cap_employee_cap FOREIGN KEY (chef_id) REFERENCES  
employees_cap(id);
```

```
CREATE SEQUENCE sequence_chefs_orders_dishes_cap  
  INCREMENT BY 2  
  START WITH 100  
  NOCYCLE;
```

```
CREATE OR REPLACE TRIGGER trigger_sequence_chefs_orders_dishes_cap  
  BEFORE INSERT ON chefs_orders_dishes_cap  
  FOR EACH ROW  
BEGIN  
  SELECT sequence_chefs_orders_dishes_cap.NEXTVAL INTO :NEW.id FROM dual;  
END;  
/
```

```
CREATE OR REPLACE TRIGGER trigger_pk_chefs_orders_dishes_cap  
  BEFORE INSERT ON chefs_orders_dishes_cap  
  FOR EACH ROW  
DECLARE  
  v_count INT;  
BEGIN  
  SELECT count(*) INTO v_count FROM chefs_orders_dishes_prov@pdb2 WHERE id = :NEW.id;  
  IF v_count <> 0 THEN  
    RAISE_APPLICATION_ERROR (-20001, 'unique constraint  
(USER_MODBD.PK_CHEFS_ORDERS_DISHES_PROV) violated');  
  END IF;  
END;  
/
```



The screenshot shows the SQL Developer interface. The top pane is the 'Worksheet' tab, displaying a SQL script for the 'chefs\_orders\_dishes' table. The script includes constraints, a sequence, and two triggers. The bottom pane is the 'Script Output' window, showing the execution results of the script.

```
1 -- chefs_orders_dishes
2 ALTER TABLE chefs_orders_dishes_cap ADD CONSTRAINT pk_chefs_orders_dishes_cap PRIMARY KEY (id);
3 ALTER TABLE chefs_orders_dishes_cap ADD CONSTRAINT fk_chefs_orders_dishes_cap_orders_cap FOREIGN KEY (order_id) REFERENCES orders_cap(id);
4 ALTER TABLE chefs_orders_dishes_cap ADD CONSTRAINT fk_chefs_orders_dishes_cap_dishes_cap FOREIGN KEY (dish_id) REFERENCES dishes_cap(id);
5 ALTER TABLE chefs_orders_dishes_cap ADD CONSTRAINT fk_chefs_orders_dishes_cap_employee_cap FOREIGN KEY (chef_id) REFERENCES employees_cap(id);
6
7 CREATE SEQUENCE sequence_chefs_orders_dishes_cap
8 INCREMENT BY 2
9 START WITH 100
10 NOCYCLE;
11
12 CREATE OR REPLACE TRIGGER trigger_sequence_chefs_orders_dishes_cap
13 BEFORE INSERT ON chefs_orders_dishes_cap
14 FOR EACH ROW
15 BEGIN
16 SELECT sequence_chefs_orders_dishes_cap.NEXTVAL INTO :NEW.id FROM dual;
17 END;
18 /
19
20 CREATE OR REPLACE TRIGGER trigger_pk_chefs_orders_dishes_cap
21 BEFORE INSERT ON chefs_orders_dishes_cap
22 FOR EACH ROW
23 DECLARE
24 v_count INT;
25 BEGIN
```

Script Output:

```
Table CHEFS_ORDERS_DISHES_CAP altered.
Table CHEFS_ORDERS_DISHES_CAP altered.
Table CHEFS_ORDERS_DISHES_CAP altered.
Table CHEFS_ORDERS_DISHES_CAP altered.
Sequence SEQUENCE_CHEFS_ORDERS_DISHES_CAP created.
Trigger TRIGGER_SEQUENCE_CHEFS_ORDERS_DISHES_CAP compiled
Trigger TRIGGER_PK_CHEFS_ORDERS_DISHES_CAP compiled
```

```
-- chefs_orders_dishes
ALTER TABLE chefs_orders_dishes_prov ADD CONSTRAINT pk_chefs_orders_dishes_prov
PRIMARY KEY (id);
ALTER TABLE chefs_orders_dishes_prov ADD CONSTRAINT
fk_chefs_orders_dishes_prov_orders_prov FOREIGN KEY (order_id) REFERENCES orders_prov(id);
ALTER TABLE chefs_orders_dishes_prov ADD CONSTRAINT
fk_chefs_orders_dishes_prov_dishes_prov FOREIGN KEY (dish_id) REFERENCES dishes_prov(id);
ALTER TABLE chefs_orders_dishes_prov ADD CONSTRAINT
fk_chefs_orders_dishes_prov_employee_prov FOREIGN KEY (chef_id) REFERENCES
employees_prov(id);
```

```
CREATE SEQUENCE sequence_chefs_orders_dishes_prov
INCREMENT BY 2
START WITH 101
NOCYCLE;
```

```
CREATE OR REPLACE TRIGGER trigger_sequence_chefs_orders_dishes_prov
BEFORE INSERT ON chefs_orders_dishes_prov
FOR EACH ROW
BEGIN
SELECT sequence_chefs_orders_dishes_prov.NEXTVAL INTO :NEW.id FROM dual;
END;
/
```

```
CREATE OR REPLACE TRIGGER trigger_pk_chefs_orders_dishes_prov
```

```

BEFORE INSERT ON chefs_orders_dishes_prov
FOR EACH ROW
DECLARE
  v_count INT;
BEGIN
  SELECT count(*) INTO v_count FROM chefs_orders_dishes_cap@pdb1 WHERE id = :NEW.id;
  IF v_count <> 0 THEN
    RAISE_APPLICATION_ERROR (-20001, 'unique constraint
(USER_MODBD.PK_CHEFS_ORDERS_DISHES_CAP) violated');
  END IF;
END;
/

```

The screenshot shows the SQL Developer interface. The top part is the 'Query Builder' window, which contains the following SQL code:

```

1  -- chefs_orders_dishes
2  ALTER TABLE chefs_orders_dishes_prov ADD CONSTRAINT pk_chefs_orders_dishes_prov PRIMARY KEY (id);
3  ALTER TABLE chefs_orders_dishes_prov ADD CONSTRAINT fk_chefs_orders_dishes_prov_orders_prov FOREIGN KEY (order_id) REFERENCES orders_prov(id);
4  ALTER TABLE chefs_orders_dishes_prov ADD CONSTRAINT fk_chefs_orders_dishes_prov_dishes_prov FOREIGN KEY (dish_id) REFERENCES dishes_prov(id);
5  ALTER TABLE chefs_orders_dishes_prov ADD CONSTRAINT fk_chefs_orders_dishes_prov_employee_prov FOREIGN KEY (chef_id) REFERENCES employees_prov(id);
6
7  CREATE SEQUENCE sequence_chefs_orders_dishes_prov
8    INCREMENT BY 2
9    START WITH 101
10   NOCYCLE;
11
12  CREATE OR REPLACE TRIGGER trigger_sequence_chefs_orders_dishes_prov
13    BEFORE INSERT ON chefs_orders_dishes_prov
14    FOR EACH ROW
15  BEGIN
16    SELECT sequence_chefs_orders_dishes_prov.NEXTVAL INTO :NEW.id FROM dual;
17  END;
18  /
19
20  CREATE OR REPLACE TRIGGER trigger_pk_chefs_orders_dishes_prov
21    BEFORE INSERT ON chefs_orders_dishes_prov
22    FOR EACH ROW
23  DECLARE
24    v_count INT;
25  BEGIN

```

The bottom part is the 'Script Output' window, which shows the execution results of the SQL code:

```

Table CHEFS_ORDERS_DISHES_PROV altered.

Table CHEFS_ORDERS_DISHES_PROV altered.

Table CHEFS_ORDERS_DISHES_PROV altered.

Table CHEFS_ORDERS_DISHES_PROV altered.

Sequence SEQUENCE_CHEFS_ORDERS_DISHES_PROV created.

Trigger TRIGGER_SEQUENCE_CHEFS_ORDERS_DISHES_PROV compiled

Trigger TRIGGER_PK_CHEFS_ORDERS_DISHES_PROV compiled

```

Am decis să adăugăm aici și generarea automata a valorilor pentru attributele price din orders\_drinks și chef\_orders\_dishes, precum și a atributelor total și order\_date din tabela orders. Trigger-ele corespunzătoare sunt create local, in pdb1 și pdb2

```

CREATE OR REPLACE TRIGGER trigger_orders_drinks_cap_price
  BEFORE INSERT OR UPDATE OR DELETE ON orders_drinks_cap
  FOR EACH ROW
DECLARE
  v_price INT;
BEGIN
  IF INSERTING THEN

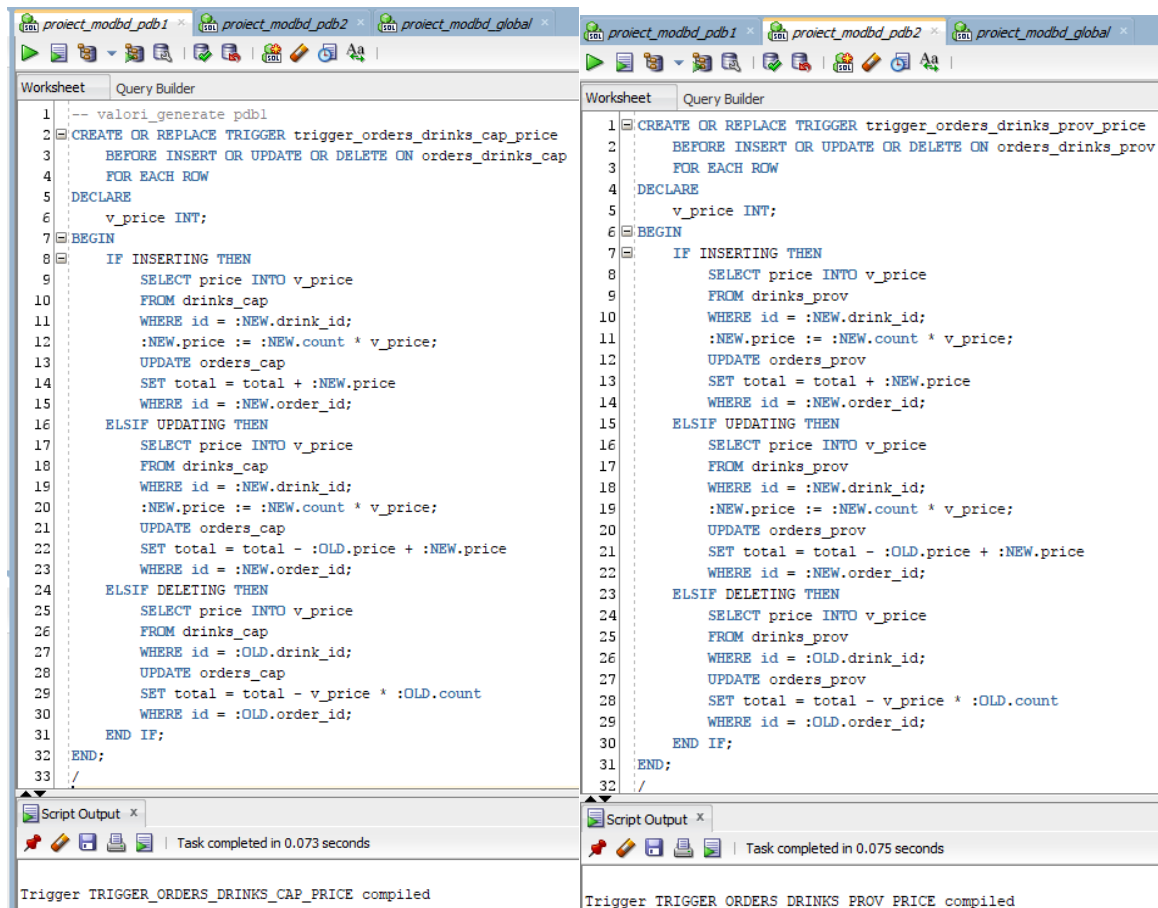
```

```

        SELECT price INTO v_price
        FROM drinks_cap
        WHERE id = :NEW.drink_id;
        :NEW.price := :NEW.count * v_price;
        UPDATE orders_cap
        SET total = total + :NEW.price
        WHERE id = :NEW.order_id;
    ELSIF UPDATING THEN
        SELECT price INTO v_price
        FROM drinks_cap
        WHERE id = :NEW.drink_id;
        :NEW.price := :NEW.count * v_price;
        UPDATE orders_cap
        SET total = total - :OLD.price + :NEW.price
        WHERE id = :NEW.order_id;
    ELSIF DELETING THEN
        SELECT price INTO v_price
        FROM drinks_cap
        WHERE id = :OLD.drink_id;
        UPDATE orders_cap
        SET total = total - v_price * :OLD.count
        WHERE id = :OLD.order_id;
    END IF;
END;
/

CREATE OR REPLACE TRIGGER trigger_orders_drinks_prov_price
    BEFORE INSERT OR UPDATE OR DELETE ON orders_drinks_prov
    FOR EACH ROW
DECLARE
    v_price INT;
BEGIN
    IF INSERTING THEN
        SELECT price INTO v_price
        FROM drinks_prov
        WHERE id = :NEW.drink_id;
        :NEW.price := :NEW.count * v_price;
        UPDATE orders_prov
        SET total = total + :NEW.price
        WHERE id = :NEW.order_id;
    ELSIF UPDATING THEN
        SELECT price INTO v_price
        FROM drinks_prov
        WHERE id = :NEW.drink_id;
        :NEW.price := :NEW.count * v_price;
        UPDATE orders_prov
        SET total = total - :OLD.price + :NEW.price
        WHERE id = :NEW.order_id;
    ELSIF DELETING THEN
        SELECT price INTO v_price
        FROM drinks_prov
        WHERE id = :OLD.drink_id;
        UPDATE orders_prov
        SET total = total - v_price * :OLD.count
        WHERE id = :OLD.order_id;
    END IF;
END;
/

```



```

CREATE OR REPLACE TRIGGER trigger_chefs_orders_dishes_cap_price
  BEFORE INSERT OR UPDATE OR DELETE ON chefs_orders_dishes_cap
  FOR EACH ROW
DECLARE
  v_price INT;
BEGIN
  IF INSERTING THEN
    SELECT price INTO v_price
    FROM dishes_cap
    WHERE id = :NEW.dish_id;
    :NEW.price := :NEW.count * v_price;
    UPDATE orders_cap
    SET total = total + :NEW.price
    WHERE id = :NEW.order_id;
  ELSIF UPDATING THEN
    SELECT price INTO v_price
    FROM dishes_cap
    WHERE id = :NEW.dish_id;
    :NEW.price := :NEW.count * v_price;
    UPDATE orders_cap
    SET total = total - :OLD.price + :NEW.price
    WHERE id = :NEW.order_id;
  ELSIF DELETING THEN
    SELECT price INTO v_price
    FROM dishes_cap
    WHERE id = :OLD.dish_id;
    UPDATE orders_cap
    SET total = total - v_price * :OLD.count
    WHERE id = :OLD.order_id;
  END IF;
END;

```

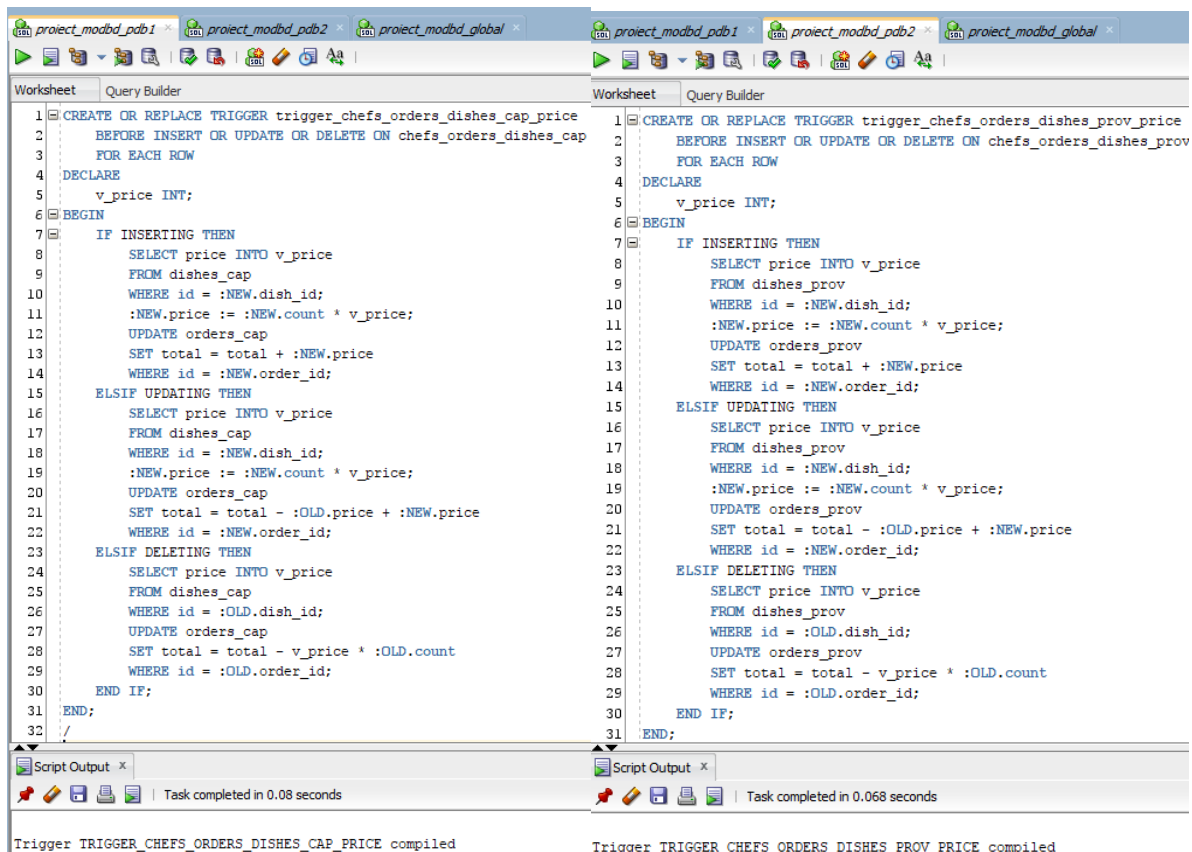
```

END;
/

CREATE OR REPLACE TRIGGER trigger_chefs_orders_dishes_prov_price
  BEFORE INSERT OR UPDATE OR DELETE ON chefs_orders_dishes_prov
  FOR EACH ROW
DECLARE
  v_price INT;
BEGIN
  IF INSERTING THEN
    SELECT price INTO v_price
    FROM dishes_prov
    WHERE id = :NEW.dish_id;
    :NEW.price := :NEW.count * v_price;
    UPDATE orders_prov
    SET total = total + :NEW.price
    WHERE id = :NEW.order_id;
  ELSIF UPDATING THEN
    SELECT price INTO v_price
    FROM dishes_prov
    WHERE id = :NEW.dish_id;
    :NEW.price := :NEW.count * v_price;
    UPDATE orders_prov
    SET total = total - :OLD.price + :NEW.price
    WHERE id = :NEW.order_id;
  ELSIF DELETING THEN
    SELECT price INTO v_price
    FROM dishes_prov
    WHERE id = :OLD.dish_id;
    UPDATE orders_prov
    SET total = total - v_price * :OLD.count
    WHERE id = :OLD.order_id;
  END IF;
END;
/

```





```

CREATE OR REPLACE TRIGGER trigger_orders_cap_date
  BEFORE INSERT OR UPDATE ON orders_cap
  FOR EACH ROW
DECLARE
  v_date DATE;
BEGIN
  IF INSERTING THEN
    SELECT sysdate INTO v_date
    FROM dual;
    :NEW.order_date := v_date;
    :NEW.total := 0;
  ELSIF UPDATING THEN
    SELECT sysdate INTO v_date
    FROM dual;
    :NEW.order_date := v_date;
  END IF;
END;
/

```

```

CREATE OR REPLACE TRIGGER trigger_orders_prov_date
  BEFORE INSERT OR UPDATE ON orders_prov
  FOR EACH ROW
DECLARE
  v_date DATE;
BEGIN
  IF INSERTING THEN
    SELECT sysdate INTO v_date
    FROM dual;
    :NEW.order_date := v_date;
    :NEW.total := 0;
  ELSIF UPDATING THEN
    SELECT sysdate INTO v_date

```

```

FROM dual;
:NEW.order_date := v_date;
END IF;
END;
/

```

Worksheet	Query Builder	Worksheet	Query Builder
<pre> 1 CREATE OR REPLACE TRIGGER trigger_orders_cap_date 2 BEFORE INSERT OR UPDATE ON orders_cap 3 FOR EACH ROW 4 DECLARE 5     v_date DATE; 6 BEGIN 7     IF INSERTING THEN 8         SELECT sysdate INTO v_date 9         FROM dual; 10        :NEW.order_date := v_date; 11        :NEW.total := 0; 12    ELSIF UPDATING THEN 13        SELECT sysdate INTO v_date 14        FROM dual; 15        :NEW.order_date := v_date; 16    END IF; 17 END; 18 / </pre>		<pre> 1 CREATE OR REPLACE TRIGGER trigger_orders_prov_date 2 BEFORE INSERT OR UPDATE ON orders_prov 3 FOR EACH ROW 4 DECLARE 5     v_date DATE; 6 BEGIN 7     IF INSERTING THEN 8         SELECT sysdate INTO v_date 9         FROM dual; 10        :NEW.order_date := v_date; 11        :NEW.total := 0; 12    ELSIF UPDATING THEN 13        SELECT sysdate INTO v_date 14        FROM dual; 15        :NEW.order_date := v_date; 16    END IF; 17 END; 18 / </pre>	
<p>Script Output x</p> <p>Task completed in 0.059 seconds</p> <p>Trigger TRIGGER_ORDERS_CAP_DATE compiled</p>		<p>Script Output x</p> <p>Task completed in 0.064 seconds</p> <p>Trigger TRIGGER_ORDERS_PROV_DATE compiled</p>	