

**UNIVERSITATEA DIN BUCUREȘTI**

FACULTATEA DE

MATEMATICĂ ȘI INFORMATICĂ

**Modalități de optimizare și distribuire în baze de date**

*Modulul proiect*

**Baze de Date și Tehnologii Software**

Cozma Laura-Elena *- Product Owner*Iamandii Ana-Maria - *Dezvoltator*

Manolache Andrei - *Dezvoltator*

*Mai 2023*

1. Descrierea modelului ales și a obiectivelor aplicației

Modelul ales în dezvoltarea aplicației de Data Warehouse reprezintă o bază de date ce conține zborurile companiilor aeriene împreună cu rezervările făcute de clienți, pentru a evidenția diverse rezultate cu privire la zborurile efectuate, destinațiile cele mai frecventate sau numărul de zboruri lunare pentru anumiți operatori de zbor.

În cadrul proiectului a fost folosit drept punct de plecare setul de date [2015 Flight Delays and Cancellations](https://www.kaggle.com/datasets/usdot/flight-delays) de pe platforma Kaggle. Acesta este populat cu date reale, ce prezintă detalii despre zborurile efectuate de companiile aeriene din Statele unite ale Americii în primele 3 luni ale anului 2015. Coloanele din tabele, precum destinația de plecare și cea de sosire, timpul programat al plecării, respectiv al sosirii, durata zborului cât și distanța efectuată, sunt colectate de către *U.S. Department of Transportation's (DOT) Bureau of Transportation Statistics*, ulterior fiind publicate în raportul lunar *Air Travel Consumer Report*.

Setul de date menționat conține următoarele tabele, sub formă de fișiere csv:

* *Airlines.csv -* conține un cod unic al companiei aeriene, împreună cu numele acesteia
* *Airports.csv -* conține codul unic al aeroportului, numele acestuia, orașul, statul, țara (fiind USA în toate cazurile) și coordonatele acestuia
* *Flights.csv -* conține data, ora de plecare, respectiv de sosire, destinația de plecare, respectiv de sosire, reprezentate prin codul aeroportului, codul liniei aeriene, numărul zborului, numărul aeronavei, durata și distanța parcursă, împreună cu alte date referitoare la întârzieri și anulări.

Conținutul tabelelor menționate este folosit în cadrul aplicației în modul următor:

* Fișierul *airlines.csv* devine sursă de date pentru tabelul OPERATOR\_ZBOR, care va conține codul operatorului, împreună cu denumirea acestuia
* Fișierul *airports.csv* definește destinațiile posibile pentru zboruri în tabelele DESTINATIE, respectiv STAT, din care vom prelua codul aeroportului, orașul și statul
* Fișierul *flights.csv* devine sursa de date pentru tabelul ZBOR, iar coloanele preluate sunt: codul operatorului, codul aeronavei, durata, distanta, coloana anulat, cu valoarea 0 dacă zborul a avut loc, respectiv 1 dacă a fost anulat, data de plecare împreună cu ora de plecare, data de sosire împreună cu ora de sosire și locația de plecare și de sosire.

În ceea ce privește tabelul AERONAVA, a fost folosit un [dataset](https://github.com/PacktPublishing/QlikView-for-Developers/blob/master/Chapter09/Aircrafts.csv?fbclid=IwAR2ZwXBfnAHfDlDc9pwyqTeXQ3J-kEZy5j-x8-pSJBsqZvXIRB_CI_SIM5U) de pe Github, din care, au fost selectate coloanele *aircraft\_name, aircraft\_short\_name* și *manufacturer.*

Deoarece fișierul *flights.csv* nu conținea toată informația pe care doream să o folosim în baza de date, am creat un script în Python care modifică fișierul csv în următorul mod:

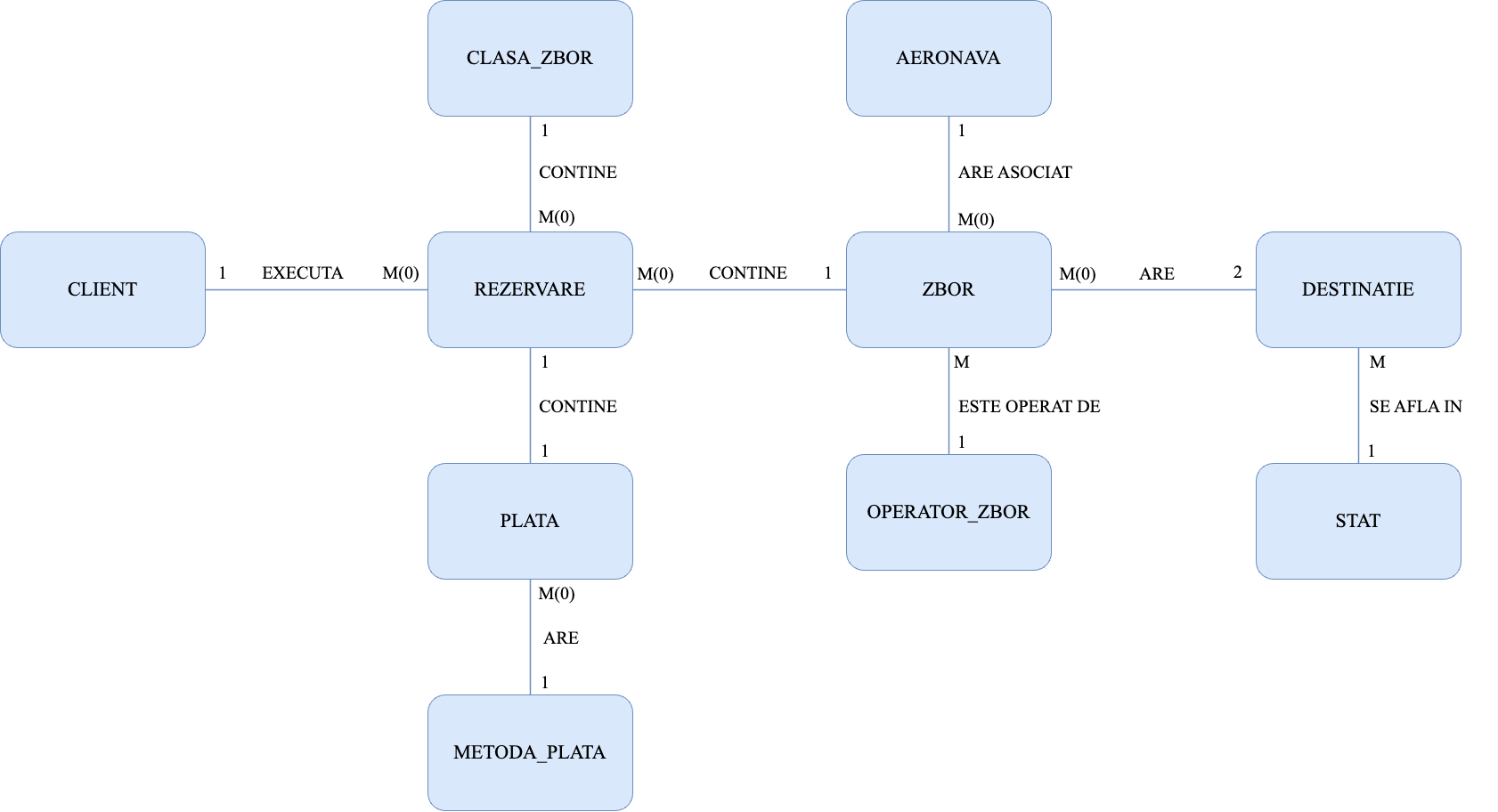
* Adaugă coloana zbor\_id, creată incremental, pentru a o folosi în continuare cu ușurintă
* Generează coloana total\_locuri, cu valori între 50 și 100
* Modifică coloanele data\_plecare și data\_sosire de tip timestamp, pentru a conține și ora
* Adaugă o cheie externă spre aeronave

Pentru a genera clienții care vor face ulterior rezervări, a fost folosit setul de date [people](https://www.datablist.com/learn/csv/download-sample-csv-files?fbclid=IwAR3mbdm566cZKftpZHzjNmcnGarOdyN0BTK1a4MkwF1iu-aAXD03b54xHaY#people-dataset) de 10000 înregistrări din care am preluat id-ul, numele, prenumele, email-ul și numărul de telefon.

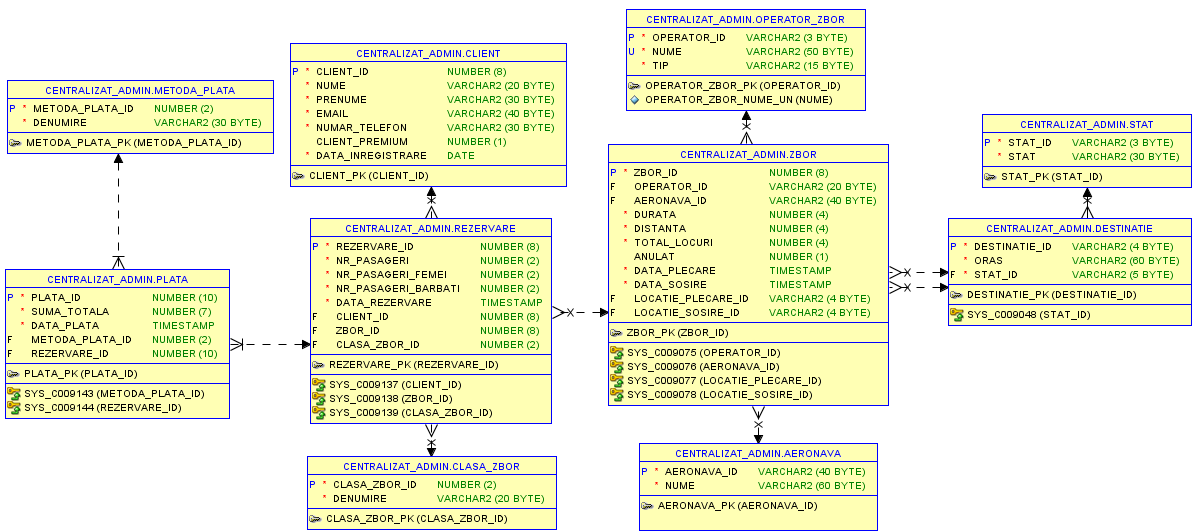
Rezervările clienților au fost generate în totalitate folosind un script de python și conțin: id-ul, numărul total de pasageri conținut de rezervare, numărul de pasageri care sunt femei, respectiv bărbați, data rezervării, id-ul clientului, id-ul zborului, clasa la care au fost rezervate locurile și id-ul plății corespunzătoare rezervării. Atât clasele de zbor, cât și metodele de plată posibile sunt stocate în tabele separate. Plățile corespunzătoare rezervărilor au fost, de asemenea, generate cu ajutorul unui script de Python.

1. Diagramele bazei de date OLTP inițiale

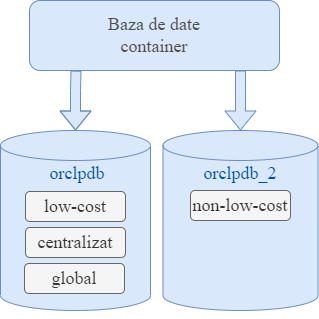
* Diagrama entitate - relație a bazei de date OLTP



* Diagrama conceptuală a bazei de date OLTP

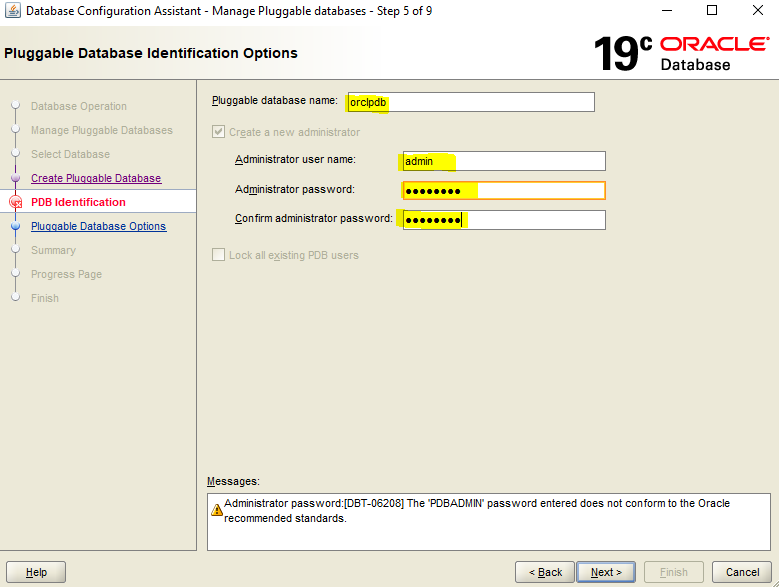
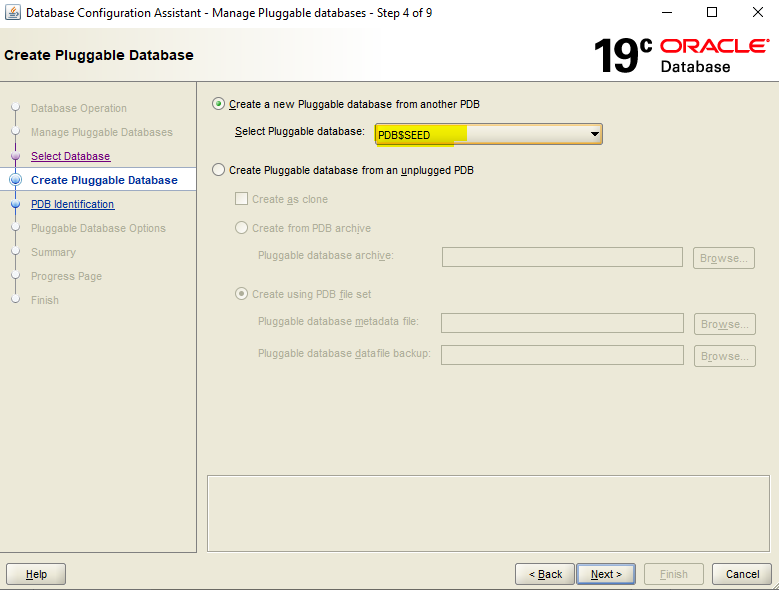
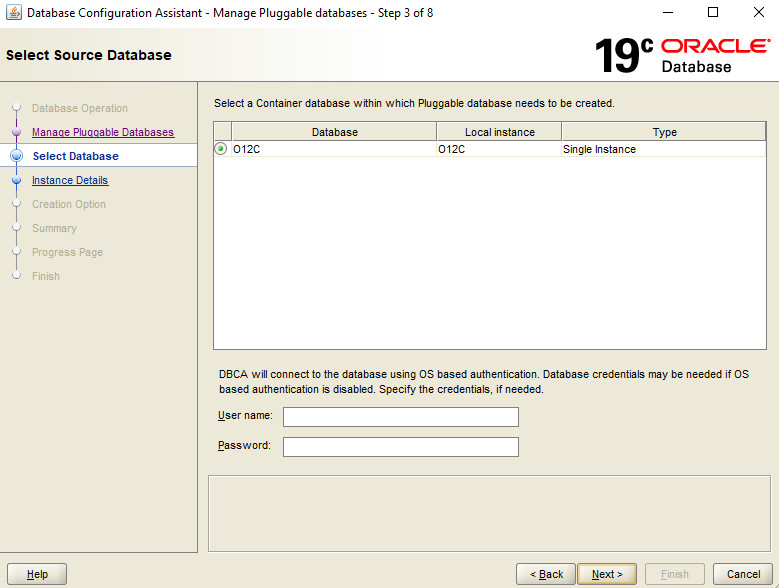
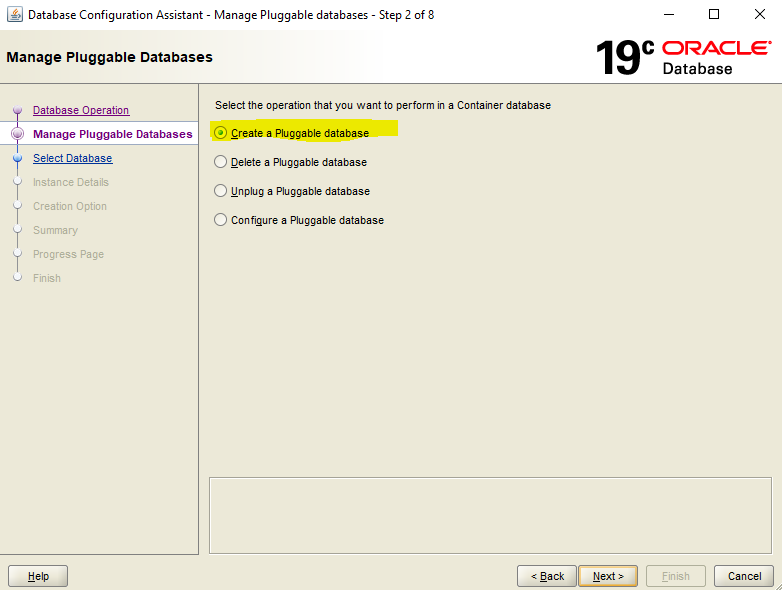


1. Descrierea modului de distribuire

 Pentru dezvoltarea proiectului, a fost folosită o bază de date *Oracle* de tip container, la care au fost adăugate două baze de date pluggable, *orclpdb* și *orclpdb\_2.* Proiectul se axează în jurul ideii de companii aeriene *low-cost* și *non-low-cost,* și este astfel distribuit pe cele două baze de date amintite anterior, *orclpdb* fiind asociată cu baza de date locală *low-cost*, iar *orclpdb\_2* cu baza de date locală *non-low-cost.* De asemenea, pentru simplitate, am folosit serverul *orclpdb* pentru stocarea obiectelor din schema utilizatorului *global*, utilizat în aplicație. Nu în ultimul rând, tot pe prima bază de date am stocat și schema *centralizat* ce conține baza de date *OLTP,* folosită pentru popularea fragmentelor.

1. Crearea bazelor de date și a utilizatorilor

* Crearea bazelor de date pluggable

În cadrul proiectului, a fost folosită o bază de date tip *container*, la care au fost adăugate două *baze de date pluggable.* Acestea au fost adăugate cu ajutorul utilitarului *Database Configuration Assistant.*

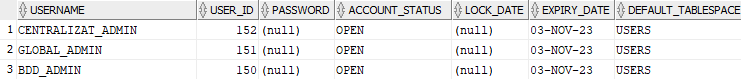
* Crearea utilizatorilor

Baza de date *orclpdb (low-cost)* conține 3 utilizatori: *centralizat\_admin*, care are în schemă tabelele inițiale din baza de date OLTP, *bdd-admin,* care va avea acces la tabelele fragmentate *low-cost* și la tabelele replicate și *global\_admin,* utilizatorul prin intermediul căruia se poate face interacțiunea în aplicație.

|  |
| --- |
| -- verificam ca ne aflam pe orclpdb SHOW con\_name; |



|  |
| --- |
| -- cream utilizatorii de pe db1 (orclpdb) CREATE USER bdd\_admin  IDENTIFIED BY bdd\_admin QUOTA UNLIMITED ON USERS;  CREATE USER global\_admin  IDENTIFIED BY global\_admin QUOTA UNLIMITED ON USERS;  CREATE USER centralizat\_admin  IDENTIFIED BY centralizat\_admin  QUOTA UNLIMITED ON USERS;  -- afisam utilizatorii creati SELECT \* FROM dba\_users ORDER BY created desc; |



Baza de date *orclpdb\_2 (non-low\_cost)* conține utilizatorul *bdd-admin*, care va avea acces la tabelele fragmentate *non-low-cost* și la tabelele replicate.

|  |
| --- |
| -- verificam ca ne aflam pe orclpdb\_2 SHOW con\_name; |

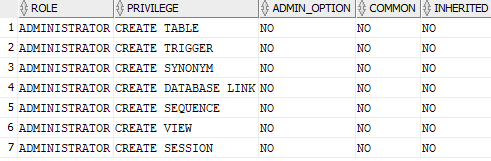


|  |
| --- |
| -- cream utilizatorul de pe db2 (orclpdb\_2) CREATE USER bdd\_admin  IDENTIFIED BY bdd\_admin QUOTA UNLIMITED ON USERS;  -- afisam utilizatorii creati SELECT \* FROM dba\_users ORDER BY created desc; |



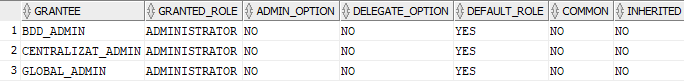
Pentru acordarea permisiunilor utilizatorilor, a fost creat *rolul administrator*, care apoi a fost atribuit fiecărui utilizator.

|  |
| --- |
| -- cream un rol de tip admin pentru a oferi drepturi utilizatorilor creati CREATE ROLE administrator; GRANT CREATE SESSION TO administrator; GRANT CREATE DATABASE LINK TO administrator; GRANT CREATE TABLE TO administrator; GRANT CREATE SEQUENCE TO administrator; GRANT CREATE SYNONYM TO administrator; GRANT CREATE VIEW TO administrator; GRANT CREATE TRIGGER TO administrator;  -- afisam rolul creat SELECT \* FROM role\_sys\_privs WHERE role = 'ADMINISTRATOR'; |



Pentru *orclpdb:*

|  |
| --- |
| -- oferim rolul utilizatorilor GRANT administrator TO bdd\_admin; GRANT administrator TO global\_admin; GRANT administrator TO centralizat\_admin;  -- verificam ca au fost aplicate rolurile SELECT \* FROM DBA\_role\_privs WHERE grantee IN ('BDD\_ADMIN', 'CENTRALIZAT\_ADMIN', 'GLOBAL\_ADMIN') ORDER BY 1; |



Pentru *orclpdb\_2:*

|  |
| --- |
| -- oferim rolul utilizatorilor GRANT administrator TO bdd\_admin;  -- verificam ca au fost aplicate rolurile SELECT \* FROM DBA\_role\_privs WHERE grantee = 'BDD\_ADMIN' ORDER BY 1; |



De asemenea, trebuie ca utilizatorii *bdd\_admin* și *global\_admin* trebuie să aibă acces în schema utilizatorului *centralizat\_admin*, pentru popularea inițială cu date. Pentru atribuirea privilegiilor a fost creată o procedură:

|  |
| --- |
| -- Userii bdd\_admin si global\_admin trebuie sa primeasca acces la tabelele din schema utilizatorului centralizat\_admin  CREATE OR REPLACE PROCEDURE grant\_select(  p\_username VARCHAR2,   p\_grantee VARCHAR2) AS   v\_cmd VARCHAR2(100); BEGIN  FOR r IN (  SELECT owner, table\_name   FROM all\_tables   WHERE owner = UPPER(p\_username)  )  LOOP  v\_cmd := 'GRANT SELECT ON '||r.owner||'.'||r.table\_name||' TO ' || p\_grantee;  EXECUTE IMMEDIATE v\_cmd;  END LOOP; END; / |

|  |
| --- |
| BEGIN  grant\_select('centralizat\_admin','bdd\_admin');  grant\_select('centralizat\_admin', 'global\_admin'); END; / |

Iar utilizatorul *global\_admin* trebuie să primească toate tipurile de acces la tabelele din schema utilizatorului *bdd\_admin*, inclusiv la secvențe, și invers:

|  |
| --- |
| -- Userul global\_admin trebuie sa primeasca toate tipurile de acces la tabelele din schema utilizatorului bdd\_admin  CREATE OR REPLACE PROCEDURE grant\_all(  p\_username VARCHAR2,   p\_grantee VARCHAR2) AS   v\_cmd VARCHAR2(100); BEGIN  FOR r IN (  SELECT owner, table\_name   FROM all\_tables   WHERE owner = UPPER(p\_username)  )  LOOP  v\_cmd := 'GRANT ALL ON '||r.owner||'.'||r.table\_name||' TO ' || p\_grantee;  EXECUTE IMMEDIATE v\_cmd;  END LOOP;    FOR r IN (  SELECT sequence\_owner, sequence\_name  FROM all\_sequences  where sequence\_owner = UPPER(p\_username)  ) LOOP  v\_cmd := 'GRANT ALL ON '||r.sequence\_owner||'.'||r.sequence\_name||' TO ' || p\_grantee;  EXECUTE IMMEDIATE v\_cmd;  END LOOP; END; / |

|  |
| --- |
| BEGIN  grant\_all('bdd\_admin','global\_admin');  grant\_all('global\_admin','bdd\_admin'); END; / |
|  |

* Crearea tabelelor din **OLTP**

AERONAVA

|  |
| --- |
| CREATE TABLE AERONAVA(  aeronava\_id varchar2(40) PRIMARY KEY,  nume VARCHAR2(60) NOT NULL ); |

STAT

|  |
| --- |
| CREATE TABLE STAT(  stat\_id VARCHAR2(3) PRIMARY KEY,  stat VARCHAR2(30) NOT NULL ); |

DESTINATIE

|  |
| --- |
| CREATE TABLE DESTINATIE  (destinatie\_id VARCHAR2(4) PRIMARY KEY,  oras VARCHAR2(60) NOT NULL,  stat\_id VARCHAR2(5) NOT NULL REFERENCES STAT(stat\_id) ON DELETE CASCADE ); |

OPERATOR\_ZBOR

|  |
| --- |
| CREATE TABLE OPERATOR\_ZBOR(  operator\_id VARCHAR2(3) PRIMARY KEY,  nume VARCHAR2(50) NOT NULL UNIQUE,  tip VARCHAR2(15) NOT NULL ); |

METODA\_PLATA

|  |
| --- |
| CREATE TABLE METODA\_PLATA(  metoda\_plata\_id NUMBER(2) PRIMARY KEY,  denumire VARCHAR2(30) NOT NULL );   CREATE SEQUENCE metoda\_plata\_seq  START WITH 1 INCREMENT BY 1;  INSERT INTO METODA\_PLATA VALUES(metoda\_plata\_seq.NEXTVAL, 'CASH'); INSERT INTO METODA\_PLATA VALUES(metoda\_plata\_seq.NEXTVAL, 'CARD'); INSERT INTO METODA\_PLATA VALUES(metoda\_plata\_seq.NEXTVAL, 'TRANSFER BANCAR'); |

CLASA\_ZBOR

|  |
| --- |
| CREATE TABLE CLASA\_ZBOR  (clasa\_zbor\_id NUMBER(2) PRIMARY KEY,  denumire VARCHAR2(20) NOT NULL );   CREATE SEQUENCE clasa\_zbor\_seq  START WITH 1 INCREMENT BY 1;  INSERT INTO CLASA\_ZBOR VALUES(clasa\_zbor\_seq.NEXTVAL, 'FIRST'); INSERT INTO CLASA\_ZBOR VALUES(clasa\_zbor\_seq.NEXTVAL, 'BUSINESS'); INSERT INTO CLASA\_ZBOR VALUES(clasa\_zbor\_seq.NEXTVAL, 'ECONOMY'); |

ZBOR

|  |
| --- |
| CREATE TABLE ZBOR(  zbor\_id NUMBER(8) PRIMARY KEY,  operator\_id VARCHAR2(20) REFERENCES OPERATOR\_ZBOR(operator\_id) ON DELETE CASCADE,  aeronava\_id VARCHAR2(40) REFERENCES AERONAVA(aeronava\_id) ON DELETE CASCADE,  durata NUMBER(4) NOT NULL,  distanta NUMBER(4) NOT NULL,  total\_locuri NUMBER(4) NOT NULL,   anulat NUMBER(1) CHECK (anulat IN (0, 1)),  data\_plecare TIMESTAMP NOT NULL,  data\_sosire TIMESTAMP NOT NULL,  locatie\_plecare\_id VARCHAR2(4) REFERENCES DESTINATIE(destinatie\_id) ON DELETE CASCADE,  locatie\_sosire\_id VARCHAR2(4) REFERENCES DESTINATIE(destinatie\_id) ON DELETE CASCADE ); |

CLIENT

|  |
| --- |
| CREATE TABLE CLIENT  (client\_id NUMBER(8) PRIMARY KEY,  nume VARCHAR2(20) NOT NULL,  prenume VARCHAR2(30) NOT NULL,  email VARCHAR2(40) NOT NULL,  numar\_telefon VARCHAR2(30) NOT NULL,  client\_premium NUMBER(1) CHECK (client\_premium IN (0, 1)),  data\_inregistrare DATE NOT NULL ); |

REZERVARE

|  |
| --- |
| CREATE TABLE REZERVARE(  rezervare\_id NUMBER(8) PRIMARY KEY,  nr\_pasageri NUMBER(2) NOT NULL CHECK(nr\_pasageri > 0),  nr\_pasageri\_femei NUMBER(2) NOT NULL CHECK(nr\_pasageri\_femei >= 0),  nr\_pasageri\_barbati NUMBER(2) NOT NULL CHECK(nr\_pasageri\_barbati >= 0),  data\_rezervare TIMESTAMP NOT NULL,  client\_id NUMBER(8) REFERENCES CLIENT(client\_id) ON DELETE CASCADE,  zbor\_id NUMBER(8) REFERENCES ZBOR(zbor\_id) ON DELETE CASCADE,  clasa\_zbor\_id NUMBER(2) REFERENCES CLASA\_ZBOR(clasa\_zbor\_id) ON DELETE CASCADE ); |

PLATA

|  |
| --- |
| CREATE TABLE PLATA  (plata\_id NUMBER(10) PRIMARY KEY,  rezervare\_id NUMBER(8) REFERENCES rezervare(rezervare\_id) ON DELETE CASCADE  suma\_totala NUMBER(7) NOT NULL,  data\_plata TIMESTAMP NOT NULL,  metoda\_plata\_id NUMBER(2) REFERENCES METODA\_PLATA(metoda\_plata\_id) ); |

Pentru popularea tabelelor s-au folosit urmatoarele script-uri în *Python:*

|  |
| --- |
| import csv import numpy as np import datetime from faker import Faker fake = Faker()   with open("plata.csv", 'w') as fout:   fout.write("plata\_id,metoda\_plata\_id,suma\_totala,data\_plata\n")   for plata\_id in range(1, 1040000):  mpi = str(np.random.choice([1,2,3]))  suma\_totala = np.random.randint(100, 10001)   start\_date = datetime.datetime(year=2015, month=1, day=1, hour=0, minute=0, second= 1)  end\_date = datetime.datetime(year=2015, month=12, day=31, hour=0, minute=0, second= 1)   data\_plata = fake.date\_between(start\_date=start\_date, end\_date=end\_date).strftime("%Y-%m-%d %H:%M:%S")  data\_plata = (datetime.datetime.strptime(data\_plata, "%Y-%m-%d %H:%M:%S")+ datetime.timedelta(minutes=np.random.randint(1,1200))).strftime("%Y-%m-%d %H:%M:%S")      fout.write(f"{plata\_id},{mpi},{suma\_totala},{data\_plata}\n")   import pdb; pdb.set\_trace() state\_id\_to\_name = {}  def get\_csv\_lines(csv\_path):  rows = []  with open(csv\_path) as fin:  csv\_reader = csv.reader(fin, delimiter=',')  line\_count = 0  for row in csv\_reader:  if line\_count == 0:  line\_count += 1   else:   rows.append(row)  line\_count +=1  return rows   for row in get\_csv\_lines("state.csv"):  state\_id\_to\_name[row[1]] = row[0]   state\_id\_to\_name["PR"] = "Puerto Rico" state\_id\_to\_name["GU"] = "Guam" state\_id\_to\_name["AS"] = "American Samoa" state\_id\_to\_name["VI"] = "Virgin Islands"    state\_counts = {}  with open("state2.csv", 'w') as fout:   fout.write("state\_id,state\n")   for destination\_row in get\_csv\_lines("destinatii.csv"):    if state\_counts.get(destination\_row[-1], 0) > 0:  continue      state\_counts[destination\_row[-1]] = state\_counts.get(destination\_row[-1], 0) + 1  fout.write(destination\_row[-1] + "," + state\_id\_to\_name[destination\_row[-1]] + "\n")    clients\_rows = [] for client\_row in get\_csv\_lines('clienti.csv'):    client\_row.append(int(np.random.randint(0, 2)))  clients\_rows.append(client\_row)   with open('clienti2.csv', 'w') as fout:    fout.write("client\_id,prenume,nume,email,numar\_telefon,client\_premium,data\_inregistrare\n")   for index, client\_row in enumerate(clients\_rows):  print(index)  start\_date = datetime.date(year=2010, month=1, day=1)  end\_date = datetime.date(year=2015, month=12, day=31)   data\_rezervare = fake.date\_between(start\_date=start\_date, end\_date=end\_date).strftime("%Y-%m-%d")  fout.write(','.join([str(x) for x in client\_row]) + f',{data\_rezervare}\n')    import pdb; pdb.set\_trace()  rows = [] with open('flights-2.csv') as csv\_file:  csv\_reader = csv.reader(csv\_file, delimiter=',')  line\_count = 0  for row in csv\_reader:  if line\_count == 0:  print(f'Column names are {", ".join(row)}')  line\_count += 1  else:  line\_count += 1   year, month, day = row[:3]  month = '0' + month if len(month) == 1 else month  day = '0' + day if len(day) == 1 else day   date = f"{year}-{month}-{day}"   rows.append([date] + row[3:])   ids = np.arange(len(rows)) + 1  for index in range(len(rows)):  rows[index].append(str(ids[index]))  rows[index].append(str(np.random.choice([50,70,80,90,100])))  datetime\_object = datetime.datetime.strptime(rows[index][0], '%Y-%m-%d') + datetime.timedelta(minutes=int(rows[index][5]))   if len(rows[index][6]) == 0:  rows[index][6] = '0'   arrival = datetime\_object + datetime.timedelta(minutes = int(rows[index][6])) if len(rows[index][6]) != 0 else datetime\_object     rows[index][5] = datetime\_object.strftime("%Y-%m-%d %H:%M:%S")  rows[index][8] = arrival.strftime("%Y-%m-%d %H:%M:%S")   rows[index] = rows[index][1:]    with open('flights\_updated.csv','w') as fout:  fout.write('operator\_id,aeronava\_id,locatie\_plecare\_id,locatie\_sosire\_id,SCHEDULED\_DEPARTURE,durata,distanta,SCHEDULED\_ARRIVAL,anulat,id\_zbor,total\_locuri\n')   for row in rows:  fout.write(','.join(row))  fout.write('\n')    import csv from pydoc import cli import numpy as np import datetime from faker import Faker fake = Faker()  rows = [] N = 1040000 for index in range(N):    rezervare\_id = index + 1  nr\_pasageri = np.random.randint(1, 11)  nr\_pasager\_femei = np.random.randint(0, nr\_pasageri + 1)  nr\_pasager\_barbati = nr\_pasageri - nr\_pasager\_femei    start\_date = datetime.datetime(year=2015, month=1, day=1, hour=0, minute=0, second= 1)  end\_date = datetime.datetime(year=2015, month=12, day=31, hour=0, minute=0, second= 1)   data\_rezervare = fake.date\_between(start\_date=start\_date, end\_date=end\_date).strftime("%Y-%m-%d %H:%M:%S")  data\_rezervare = (datetime.datetime.strptime(data\_rezervare, "%Y-%m-%d %H:%M:%S")+ datetime.timedelta(minutes=np.random.randint(1,1200))).strftime("%Y-%m-%d %H:%M:%S")   suma\_totala = np.random.randint(100, 10001)   client\_id = np.random.randint(1, 10001)   zbor\_id = np.random.randint(1, 1048575 + 1)   clasa\_id = np.random.randint(1, 4)    plata = np.random.randint(1, 4)   rows.append([str(rezervare\_id), str(nr\_pasageri), str(nr\_pasager\_femei), str(nr\_pasager\_barbati), data\_rezervare, str(client\_id), str(zbor\_id), str(clasa\_id), str(rezervare\_id)])  # rows.append([str(rezervare\_id), str(nr\_pasageri), str(nr\_pasager\_femei), str(nr\_pasager\_barbati), data\_rezervare, str(suma\_totala), str(client\_id), str(zbor\_id), str(clasa\_id), str(plata)])    with open('reservation.csv','w') as fout:  fout.write('rezervare\_id,nr\_pasageri,nr\_pasageri\_femei,nr\_pasageri\_barbati,data\_rezervare,client\_id,zbor\_id,clasa\_id,plata\_id\n')   for row in rows:  fout.write(','.join(row))  fout.write('\n')  aeronava\_ids = [row for row in get\_csv\_lines("temp2.csv")]  IDS = [x[-1] for x in aeronava\_ids]  with open('temp2.csv', 'w') as fout:    fout.write('Manufacturer,Aircraft Name,Aeronava\_ID\n')  for row in aeronava\_ids:   fout.write(','.join(row[2:]) + '\n')   zbor\_rows = get\_csv\_lines("zbor.csv")  with open("zbor2.csv", 'w') as fout:  fout.write("operator\_id,aeronava\_id,locatie\_plecare\_id,locatie\_sosire\_id,data\_plecare,durata,distanta,data\_sosire,anulat,zbor\_id,total\_locuri\n")   for row in zbor\_rows:   row[1] = str(np.random.choice(IDS))  fout.write(','.join(row) + '\n')   import pdb; pdb.set\_trace() |

1. Fragmentarea relațiilor

După cum a fost menționat anterior, a fost aleasă fragmentarea orizontală primară pornind de la tabela OPERATOR\_ZBOR, în OPERATOR\_ZBOR\_LOWCOST și OPERATOR\_ZBOR\_NON\_LOWCOST. În continuare, au rezultat fragmentările orizontale derivate, asupra tabelelor ZBOR, REZERVARE și PLATI.

În ceea ce privește fragmentarea verticală, datele clienților au fost împărțite în date GDPR, sensibile, care trebuie stocate pe un server cu o securitate mai ridicată, unde au acces numai anumiți utilizatori, și datele NON-GDPR, care pot rămâne în bazele de date locale. Pentru stocarea bazelor de date GDPR am folosit schema utilizatorului *global*, pentru a nu mai crea o nouă bază de date pluggable pentru aceasta.

* Fragmentarea orizontală primară

Algoritmul de fragmentare orizontală primară:

Se consideră relația OPERATOR\_ZBOR.

Fie Pr = {p1, p2} mulțimea de predicate simple, unde p1 și p2 reprezintă:

p1: tip = *Low cost*

p2: tip = *Non low cost*

Domeniul pentru atributul *tip* este {*Low cost, Non low cost*}. Astfel, mulțimea va conține două implicații:

i1: (tip = *Low cost*)⇒ ⏋(tip = *Non low cost*)

i2: ⏋(tip = *Low cost*)⇒ (tip = *Non low cost*)

Din acestea rezultă că dacă un operator zbor are tipul *Low cost,* atunci în mod sigur nu va putea avea valoarea *Non low cost.* De asemenea, dacă tipul unui operator zbor nu este *Low cost,* atunci știm ca va fi *Non low cost.*

Folosind combinații booleene se formează următoarele predicate compuse, ce compun mulțimea M = {m1, m2, m3, m4}:

m1: tip = *Low cost* ⌃ tip = *Non low cost*

m2: tip = *Low cost* ⌃ ⏋(tip = *Non low cost*)

m3: ⏋(tip = *Low cost*)⌃ tip = *Non low cost*

m4: ⏋(tip = *Low cost*)⌃ ⏋(tip = *Non low cost*)

Se observă că predicatele compuse m1 și m4  nu au sens în raport cu implicațiile definite în mulțimea I, astfel că trebuie eliminate din mulțimea M.

În concluzie, mulțimea predicatelor compuse care determină fragmentele orizontale ale relației este formată din {m2și m3}. Deci, relația OPERATOR\_ZBOR va fi formată din două fragmente, cea corespunzătoare operatorilor de zbor *Low cost* și cea a operatorilor de zbor *Non low cost.*

* Fragmentarea orizontală derivată

Pornind de la fragmentarea orizontală primară a relației OPERATOR\_ZBOR, construim fragmentările orizontale derivate.

Fie legătura L dintre relațiile OPERATOR\_ZBOR și ZBOR:

*owner(L) = OPERATOR\_ZBOR*

*member(L) = ZBOR*

În funcție de operatorul de zbor, zborurile pot fi grupate în două fragmente: zborurile operate de operatori *low cost* și zborurile operate de operatori *non low cost.* Tabelele pot fi definite în modul următor:

ZBORlow-cost = ZBOR ⋉ OPERATOR\_ZBORlow-cost

ZBORnon-low-cost = ZBOR ⋉ OPERATOR\_ZBORnon-low-cost

Construim următoarele fragmente orizontale derivate. Fie legătura L dintre relațiile ZBOR și REZERVARE:

*owner(L) = ZBOR*

*member(L) = REZERVARE*

În funcție de tipul zborului, rezervările pot fi pentru zboruri de tip *low-cost* și pentru zboruri de tip *non-low-cost.* Cele două fragmente corespunzătoare relației rezervare sunt definite astfel:

REZERVARElow-cost = REZERVARE ⋉ ZBORlow-cost

REZERVAREnon-low-cost = REZERVARE ⋉ ZBORnon-low-cost

Nu în ultimul rând, avem legătura L dintre relațiile REZERVARE și PLATA:

*owner(L) = REZERVARE*

*member(L) = PLATA*

În funcție de tipul rezervării, plățile corespunzătoare acestora pot fi pentru rezervări *low-cost* sau *non-low-cost.*

PLATAlow-cost = PLATA ⋉ REZERVARE low-cost

PLATAnon-low-cost = PLATA ⋉ REZERVARE non-low-cost

* Fragmentarea verticală

Algoritmul de fragmentare verticală

Se considera relația *CLIENT(client\_id, nume, prenume, email, numar\_telefon, client\_premium, data\_inregistrare)*.

Presupunem că pe această relație sunt definite următoarele aplicații, a căror specificație este dată în SQL:

* q1: găsește numele, prenumele și email-ul clientilor cu adrese de gmail.

|  |
| --- |
| SELECT nume, prenume, email FROM client WHERE email LIKE ‘%@gmail%’; |

* q2: afișează numerele de telefon si adresele de email ale tuturor clienților

|  |
| --- |
| SELECT email, numar\_telefon  FROM client; |

* q3: afișați numărul de clienți premium a căror data\_înregistrare a avut loc după o anumită dată

|  |
| --- |
| SELECT SUM(premium) FROM client WHERE data\_inregistrare >= TO\_DATE('01-JAN-21', 'DD-MON-YY'); |

* q4: Afișează nume, prenumele si daca clientul e premium sau nu, pentru toti clientii.

|  |
| --- |
| SELECT nume, prenume, premium FROM client; |

În relație cu aceste aplicații, putem defini valorile folosirii atributele relației CLIENT. Pentru aceasta, vom atribui atributelor următoarele notații:

A1 = nume

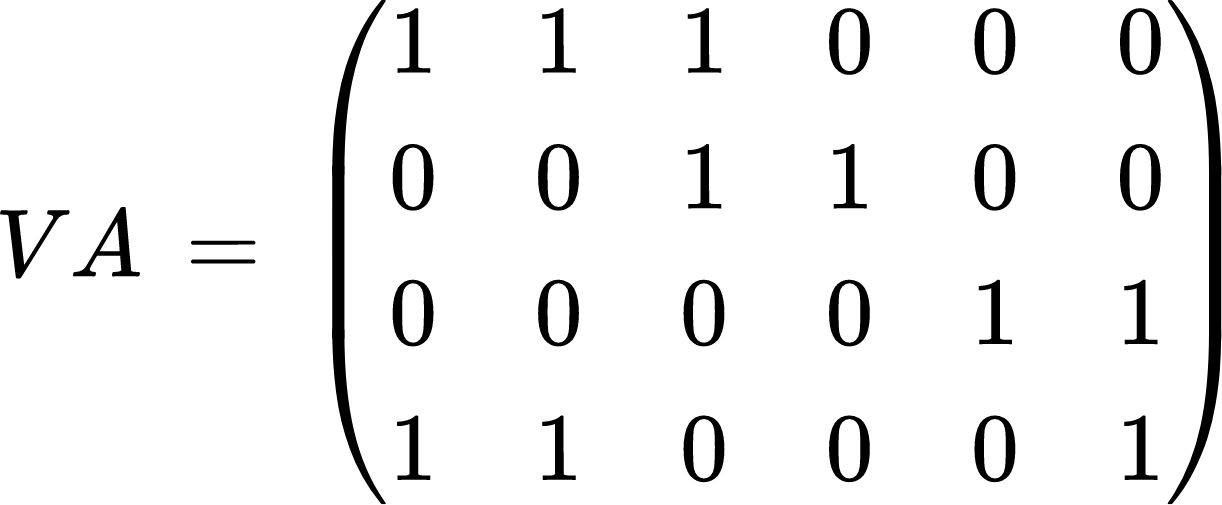
A2 = prenume

A3 = email

A4 = numar\_telefon

A6 = data\_inregistrare

A5 = premium

Valorile folosirii atributelor sunt definite în matricea VA, care va avea valorile:

Matricea VA nu conține și numărul de accesări și frecvența cu care sunt accesate aplicațiile enunțate mai sus. Rețeaua va fi formată din două stații. Am presupus ca numărul de accesări este 1 pentru toate stațiile pentru toate cererile.

*nr\_accl(qk) = 1,* pentru orice k din {1, 2, 3, 4} și l din {1, 2}

Astfel că vom presupune că cererile au următoarele frecvențe de acces:

|  |  |  |
| --- | --- | --- |
|  | Stația 1 | Stația 2 |
| fr\_acc1(q1) | 25 | 30 |
| fr\_acc1(q2) | 15 | 10 |
| fr\_acc1(q3) | 30 | 20 |
| fr\_acc1(q4) | 5 | 5 |

Calculăm valorile afinităților:

af(A1, A1) = fr\_acc1(q1) + fr\_acc2(q1) + fr\_acc1(q4) + fr\_acc2(q4) = 25 + 30 + 5 + 5 = 65

af(A1, A2) = fr\_acc1(q1) + fr\_acc2(q1) + fr\_acc1(q4) + fr\_acc2(q4) = 25 + 30 + 5 + 5 = 65

af(A1, A3) = fr\_acc1(q1) + fr\_acc2(q1) = 25 + 30 = 55

af(A1, A4) = 0

af(A1, A5) = 0

af(A1, A6) = fr\_acc1(q4) + fr\_acc2(q4) = 5 + 5 = 10

af(A2, A2) = fr\_acc1(q1) + fr\_acc2(q1) + fr\_acc1(q4) + fr\_acc2(q4) = 25 + 30 + 5 + 5 = 65

af(A2, A3) = fr\_acc1(q1) + fr\_acc2(q1) = 25 + 30 = 55

af(A2, A4) = 0

af(A2, A5) = 0

af(A2, A6) = fr\_acc1(q4) + fr\_acc2(q4) = 5 + 5 = 10

af(A3, A3) = fr\_acc1(q1) + fr\_acc2(q1) + fr\_acc1(q2) + fr\_acc2(q2) = 25 + 30 + 15 + 10 = 80

af(A3, A4) = fr\_acc1(q2) + fr\_acc2(q2) = 15 + 10 = 25

af(A3, A5) = 0

af(A3, A6) = 0

af(A4, A4) = fr\_acc1(q2) + fr\_acc2(q2) = 15 + 10 = 25

af(A4, A5) = 0

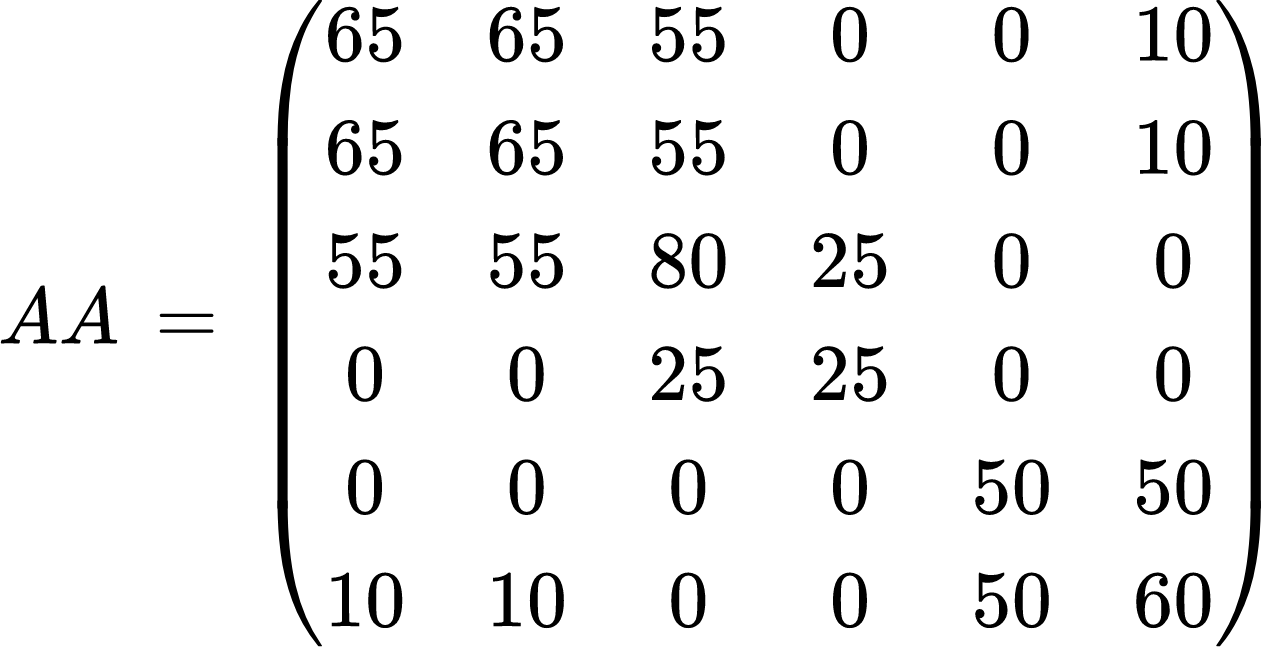
af(A4, A6) = 0

af(A5, A5) = fr\_acc1(q3) + fr\_acc2(q3) = 30 + 20 = 50

af(A5, A6) = fr\_acc1(q3) + fr\_acc2(q3) = 30 + 20 = 50

af(A6, A6) = fr\_acc1(q3) + fr\_acc2(q3) + fr\_acc1(q4) + fr\_acc2(q4) = 30 + 20 + 5 + 5 = 60

Matricea afinității atributelor este următoarea:



Calculăm contribuțiile plasărilor atributelor. Fixam atributele A1 si A2.

Plasăm A3:

cont(A0, A3, A1) = 2bond(A0, A3) + 2bond(A3, A1) - 2bond(A0, A1) = 2 \* 11550 = 23100

cont(A1, A3, A2) = 2bond(A1, A3) + 2bond(A3, A2) - 2bond(A1, A2) = 2 \* 11550 + 2 \* 11550 - 2 \* 11575 = 23050

cont(A2, A3, A0) = 2bond(A2, A3) + 2bond(A3, A0) - 2bond(A2, A0) = 2 \* 11550 = 23100

bond(A1, A2) = 65 \* 65 + 65 \* 65 + 55 \* 55 + 10 \* 10 = 11575

bond(A1, A3) = 65 \* 55 + 65 \* 55 + 55 \* 80 = 11550

bond(A2, A3) = 65 \* 55 + 65 \* 55 + 55 \* 80 = 11550

Conform calculator, putem plasa A3 inainte de A1 sau dupa A2. Il vom plasa dupa A2.

Momentan, avem ordinea A1, A2, A3.

Calculăm plasarea lui A4.

cont(A0, A4, A1) = 2bond(A0, A4) + 2bond(A4, A1) - 2bond(A0, A1) = 2 \* 1375 = 2750

cont(A1, A4, A2) = 2bond(A1, A4) + 2bond(A4, A2) - 2bond(A2, A1) = 2 \* 1375 + 2 \* 1375 - 2 \* 11575 = -17650

cont(A2, A4, A3) = 2bond(A2, A4) + 2bond(A4, A3) - 2bond(A3, A2) = 2 \* 1375 + 2 \* 2625 - 2 \* 11550 = -15100

cont(A3, A4, A0) = 2bond(A3, A4) + 2bond(A4, A0) - 2bond(A3, A0) = 2 \* 2625 = 5250

bond(A1, A4) = 55 \* 25 = 1375

bond(A2, A4) = 55 \* 25 = 1375

bond(A3, A4) = 80 \* 25 + 25 \* 25 = 2625

Cea mai buna varianta este după A4. Vom avea ordinea: A1, A2, A3, A4.

Calculăm plasarea lui A5.

cont(A0, A5, A1) = 2bond(A0, A5) + 2bond(A5, A1) - 2bond(A0, A1) = 2 \* 500 = 1000

cont(A1, A5, A2) = 2bond(A1, A5) + 2bond(A5, A2) - 2bond(A1, A2) = 2 \* 500 + 2 \* 500 - 2 \* 11575 = -21150

cont(A2, A5, A3) = 2bond(A2, A5) + 2bond(A5, A3) - 2bond(A2, A3) = 2 \* 500 + 0 - 2 \* 11550 = -22100

cont(A3, A5, A4) = 2bond(A3, A5) + 2bond(A5, A4) - 2bond(A3, A4) = - 2 \* 2625 = -5250

cont(A4, A5, A0) = 2bond(A4, A5) + 2bond(A5, A0) - 2bond(A4, A0) = 0

bond(A1, A5) = 10 \* 50 = 500

bond(A2, A5) = 10 \* 50 = 500

bond(A3, A5) = 0

bond(A4, A5) = 0

Il vom plasa pe A5 la inceput. Rezulta asezarea A5, A1, A2, A3, A4.

Calculam plasarea lui A6.

cont(A0, A6, A5) = 2bond(A0, A6) + 2bond(A6, A5) - 2bond(A0, A5) = 2 \* 5500 = 11000

cont(A5, A6, A1) = 2bond(A5, A6) + 2bond(A6, A1) - 2bond(A1, A5) = 2 \* 5500 + 2 \* 1900 - 2 \* 500 = 13800

cont(A1, A6, A2) = 2bond(A1, A6) + 2bond(A6, A2) - 2bond(A2, A1) = 2 \* 1900 + 2 \* 1900 - 2 \* 11575 = -15550

cont(A2, A6, A3) = 2bond(A2, A6) + 2bond(A6, A3) - 2bond(A3, A2) = 2 \* 1900 + 2 \* 1100 - 2 \* 11550 = -17100

cont(A3, A6, A4) = 2bond(A3, A6) + 2bond(A6, A4) - 2bond(A3, A4) = 2 \* 1100 + 0 - 2 \* 2625 = -3050

cont(A4, A6, A0) = 2bond(A4, A6) + 2bond(A6, A0) - 2bond(A4, A0) = 0

bond(A1, A6) = 65 \* 10 + 65 \* 10 + 10 \* 60 = 1900

bond(A2, A6) = 65 \* 10 + 65 \* 10 + 10 \* 60 = 1900

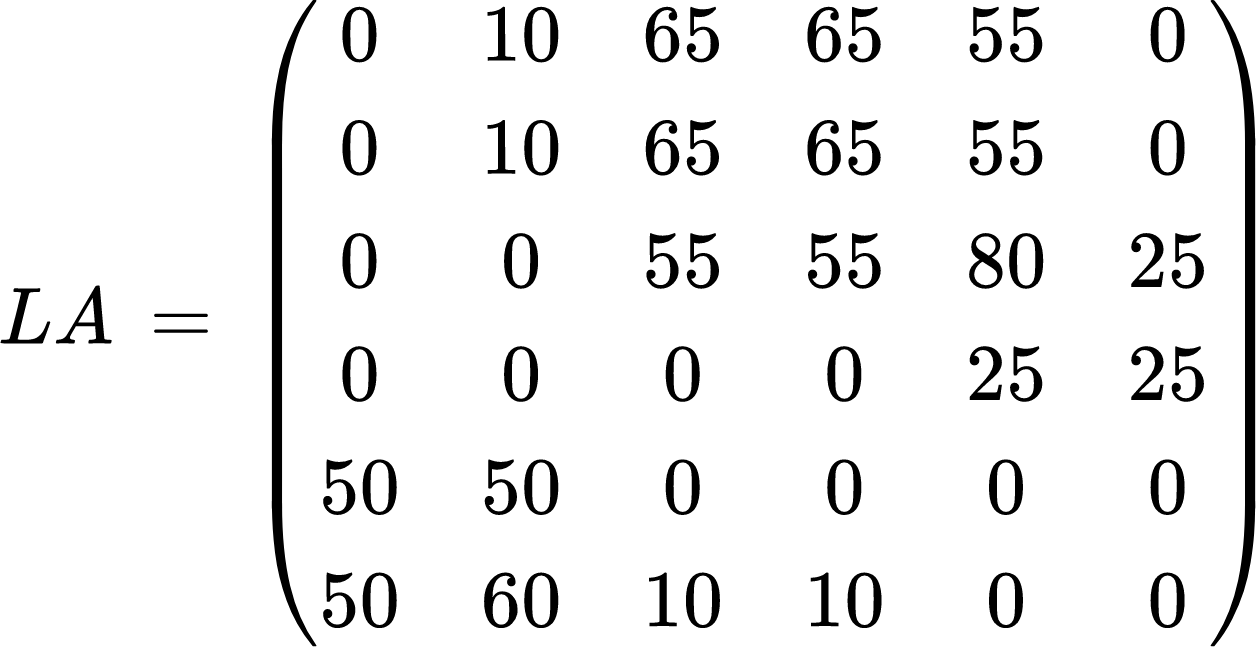
bond(A3, A6) = 55 \* 10 + 55 \* 10 = 1100

bond(A4, A6) = 0

bond(A5, A6) = 50 \* 50 + 50 \* 60 = 5500

Deci, vom plasa A6 intre A5 si A1.

Ordinea finala este: A5, A6, A1, A2, A3, A4.

Construim matricea legăturilor de afinitate:

Aflăm punctul de divizare.

Q = {q1, q2, q3, q4}

n = 1

TA1 = {A5}

BA1 = {A6, A1, A2, A3, A4}

TQ1 = ∅ => CTQ1= 0

BQ1 = {q1, q2, a4} => CBQ1 = fr\_acc1(q1) + fr\_acc2(q1) + fr\_acc1(q2) + fr\_acc2(q2) + fr\_acc1(q4) + fr\_acc2(q4) = 25 + 30 + 15 + 10 + 5 + 5 = 90

OQ1 = {q3} => COQ1 = fr\_acc1(q3) + fr\_acc2(q3) = 50

z1 = CTQ1\*CBQ1 - COQ12 = -2500

n = 2

TA2 = {A5, A6}

BA2 = {A1, A2, A3, A4}

TQ2 = {q3} => CTQ2= fr\_acc1(q3) + fr\_acc2(q3) = 30 + 20 = 50

BQ2 = {q1, q2} => CBQ2 = fr\_acc1(q1) + fr\_acc2(q1) + fr\_acc1(q2) + fr\_acc2(q2) = 25 + 30 + 15 + 10 = 80

OQ2 = {q4} => COQ2 = fr\_acc1(q4) + fr\_acc2(q4) = 5 + 5 = 10

z2 = CTQ2\*CBQ2 - COQ22 = 50 \* 80 - 100 = 3900

n = 3

TA3 = {A5, A6, A1}

BA3 = {A2, A3, A4}

TQ3 = {q3} => CTQ3= fr\_acc1(q3) + fr\_acc2(q3) = 30 + 20 = 50

BQ3 = {q2} => CBQ3 = fr\_acc1(q2) + fr\_acc2(q2) = 15 + 10 = 25

OQ3 = {q1, q4} => COQ3 = fr\_acc1(q1) + fr\_acc2(q1) + fr\_acc1(q4) + fr\_acc2(q4) = 25 + 30 + 5 + 5 = 65

z3 = CTQ3\*CBQ3 - COQ32 = 50 \* 25 - 4225 = -2975

n = 4

TA4 = {A5, A6, A1, A2}

BA4 = {A3, A4}

TQ4 = {q3, q4} => CTQ4= fr\_acc1(q3) + fr\_acc2(q3) + fr\_acc1(q4) + fr\_acc2(q4) = 30 + 20 + 5 + 5 = 60

BQ4 = {q2} => CBQ4 = fr\_acc1(q2) + fr\_acc2(q2) = 15 + 10 = 25

OQ4 = {q1} => COQ4 = fr\_acc1(q1) + fr\_acc2(q1) = 25 + 30 = 55

z4 = CTQ4\*CBQ4 - COQ42 = 60 \* 25 - 3025 = 1525

n = 5

TA5 = {A5, A6, A1, A2, A3}

BA5 = {A4}

TQ5 = {q1, q3, q4} => CTQ5= fr\_acc1(q1) + fr\_acc2(q1) + fr\_acc1(q3) + fr\_acc2(q3) + fr\_acc1(q4) + fr\_acc2(q4) = 25 + 30 + 30 + 20 + 5 + 5 = 115

BQ5 = ∅ => CBQ5 = 0

OQ5 = {q2} => COQ5 = fr\_acc1(q2) + fr\_acc2(q2) = 15 + 10 = 25

z5 = CTQ5\*CBQ5 - COQ52 = 0 - 625 = - 625

Se observă că z2 are valoarea cea mai mare, 3900, astfel că putem construi următoarele două fragmente:

CLIENT1(client\_id, data\_inregistrare, premium)

CLIENT2(client\_id, nume, prenume, email, numar\_telefon)

1. Crearea relațiilor și a fragmentelor

Crearea relațiilor

La crearea legăturilor a fost luat în calcul că se dorește o securitate mai mare, astfel că s-a ales un *database link* de tip *private.* De asemenea, a fost aleasă varianta *connected to … identified by* pentru a asocia legătura cu un anumit utilizator.

Au fost create următoarele relații între bazele de date:

Din *orclpdb (low-cost)*

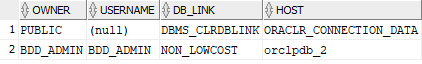
* Legătură către *bdd\_admin* din non-*lowcost* din *bdd\_admin* din *lowcost*

|  |
| --- |
| CREATE DATABASE LINK non\_lowcost CONNECT TO bdd\_admin IDENTIFIED BY bdd\_admin  USING 'orclpdb\_2'; |

Initial comanda nu va functiona. Trebuie adaugat în fișierul *$ORACLE\_HOME/network/admin/tnsnames.ora*:

|  |
| --- |
| ORCLPDB\_2 =  (DESCRIPTION =  (ADDRESS = (PROTOCOL = TCP)(HOST = localhost)(PORT = 1521))  (CONNECT\_DATA =  (SERVER = DEDICATED)  (SERVICE\_NAME = orclpdb\_2)  )) |

|  |
| --- |
| -- legaturile de baze de date la care are acces utilizatorul conectat SELECT OWNER, USERNAME, DB\_LINK, HOST FROM ALL\_DB\_LINKS; |



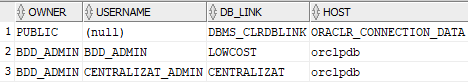
Din *orclpdb\_2 (non-lowcost)*

* Legătură către *bdd\_admin* din *lowcost* pentru utilizatorul *bdd\_admin* din *non-lowcost*

|  |
| --- |
| CREATE DATABASE LINK lowcost CONNECT TO bdd\_admin IDENTIFIED BY bdd\_admin  USING 'orclpdb'; |

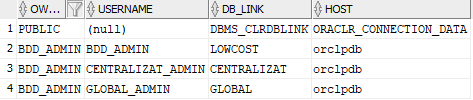
* Legătură către *centralizat\_admin* din *lowcost* pentru utilizatorul *bdd\_admin* din *non-lowcost*. Accesul către tabelele din *centralizat* se putea realiza și prin schema *bdd\_admin*

|  |
| --- |
| CREATE DATABASE LINK centralizat CONNECT TO centralizat\_admin IDENTIFIED BY centralizat\_admin USING 'orclpdb'; |



* Legătură către *global\_admin* din *lowcost* pentru utilizatorul *bdd\_admin* din *non-lowcost*

|  |
| --- |
| CREATE DATABASE LINK global CONNECT TO global\_admin IDENTIFIED BY global\_admin USING 'orclpdb'; |



Din *global:*

* Legătură către *bdd\_admin* din *non-lowcost,* din *global\_admin*.

|  |
| --- |
| CREATE DATABASE LINK non\_lowcost CONNECT TO bdd\_admin IDENTIFIED BY bdd\_admin  USING 'orclpdb\_2'; |



Crearea fragmentelor

În baza de date *low-cost:*

OPERATOR\_ZBOR\_LOWCOST

|  |
| --- |
| CREATE TABLE OPERATOR\_ZBOR\_LOWCOST  (operator\_id VARCHAR2(3),  nume VARCHAR2(50) ,  tip VARCHAR2(15) ); |

ZBOR\_LOWCOST

|  |
| --- |
| CREATE TABLE ZBOR\_LOWCOST(  zbor\_id NUMBER(8),  operator\_id VARCHAR2(20),  aeronava\_id VARCHAR2(40),  durata NUMBER(4),  distanta NUMBER(4),  total\_locuri NUMBER(4),   anulat NUMBER(1),  data\_plecare TIMESTAMP,  data\_sosire TIMESTAMP,  locatie\_plecare\_id VARCHAR2(4),  locatie\_sosire\_id VARCHAR2(4) ); |

REZERVARE\_LOWCOST

|  |
| --- |
| CREATE TABLE REZERVARE\_LOWCOST(  rezervare\_id NUMBER(8),  nr\_pasageri NUMBER(2),  nr\_pasageri\_femei NUMBER(2),  nr\_pasageri\_barbati NUMBER(2),  data\_rezervare TIMESTAMP,  client\_id NUMBER(8),  zbor\_id NUMBER(8),  clasa\_zbor\_id NUMBER(2) ); |

PLATA\_LOWCOST

|  |
| --- |
| CREATE TABLE PLATA\_LOWCOST  (plata\_id NUMBER(10),  suma\_totala NUMBER(7),  data\_plata TIMESTAMP,  metoda\_plata\_id NUMBER(2),  rezervare\_id NUMBER(10) ); |

CLIENT\_NONGDPR

|  |
| --- |
| CREATE TABLE CLIENT\_NONGDPR(  client\_id NUMBER(8),  premium NUMBER,  data\_inregistrare DATE ); |

În cazul bazei de date *Non-lowcost* crearea tabelelor e similară.

În global:

CLIENT\_GDPR

|  |
| --- |
| CREATE TABLE CLIENT\_GDPR  (client\_id NUMBER(8) PRIMARY KEY,  nume VARCHAR2(20) NOT NULL,  prenume VARCHAR2(30) NOT NULL,  email VARCHAR2(40) NOT NULL,  numar\_telefon VARCHAR2(30) NOT NULL ); |

Crearea tabelelor replicate (tabelele au aceeași structură indiferent de baza de date):

AERONAVA

|  |
| --- |
| CREATE TABLE AERONAVA(  aeronava\_id varchar2(40),  nume VARCHAR2(60) ); |

STAT

|  |
| --- |
| CREATE TABLE STAT(  stat\_id VARCHAR2(3),  stat VARCHAR2(30) ); |

DESTINATIE

|  |
| --- |
| CREATE TABLE DESTINATIE  (destinatie\_id VARCHAR2(4),  oras VARCHAR2(60),  stat\_id VARCHAR2(5)  ); |

METODA\_PLATA

|  |
| --- |
| CREATE TABLE METODA\_PLATA  (metoda\_plata\_id NUMBER(2),  denumire VARCHAR2(30) ); |

CLASA ZBOR

|  |
| --- |
| CREATE TABLE CLASA\_ZBOR  (clasa\_zbor\_id NUMBER(2),  denumire VARCHAR2(20) ); |

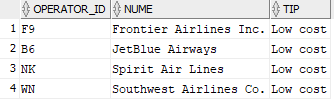
1. Popularea cu date a bazelor de date

* Popularea tabelelor fragmentate
  + Fragmentarea orizontală

În *lowcost:*

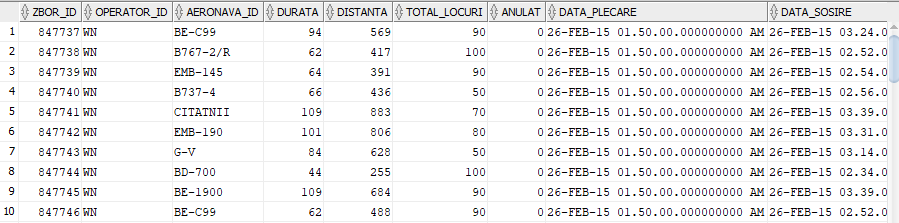
OPERATOR\_ZBOR\_LOWCOST

|  |
| --- |
| INSERT INTO operator\_zbor\_lowcost SELECT \* FROM centralizat\_admin.operator\_zbor WHERE tip = 'Low cost'; |

**

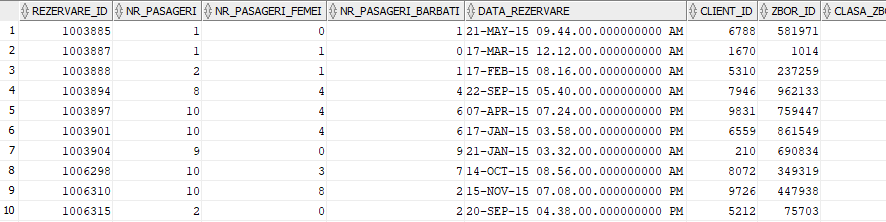
ZBOR\_LOWCOST

|  |
| --- |
| -- creare fragment zbor\_lowcost INSERT INTO zbor\_lowcost SELECT \* FROM centralizat\_admin.zbor z WHERE EXISTS (SELECT 1 FROM operator\_zbor\_lowcost o WHERE z.operator\_id = o.operator\_id); |



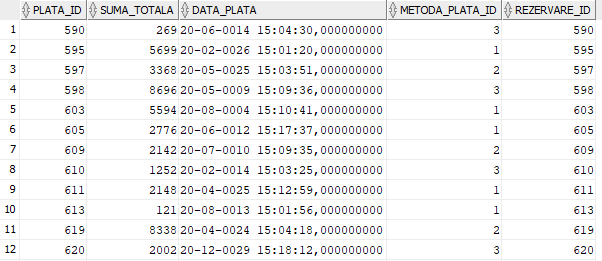
REZERVARE\_LOWCOST

|  |
| --- |
| INSERT INTO rezervare\_lowcost SELECT \* FROM centralizat\_admin.rezervare r WHERE EXISTS (SELECT 1 FROM zbor\_lowcost z WHERE r.zbor\_id = z.zbor\_id); |



PLATA\_LOWCOST

|  |
| --- |
| INSERT INTO plata\_lowcost SELECT p.\* FROM centralizat\_admin.plata p JOIN rezervare\_lowcost r ON (r.rezervare\_id = p.rezervare\_id); |



În *non-lowcost:*

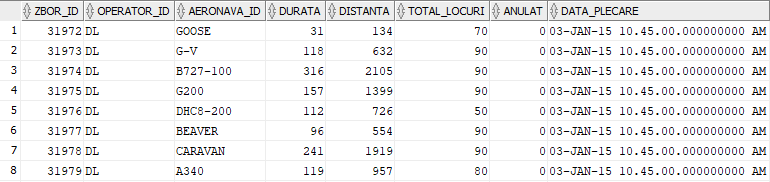
OPERATOR\_ZBOR\_NONLOWCOST

|  |
| --- |
| INSERT INTO operator\_zbor\_nonlowcost SELECT \* FROM operator\_zbor@centralizat WHERE tip = 'Non low cost'; |



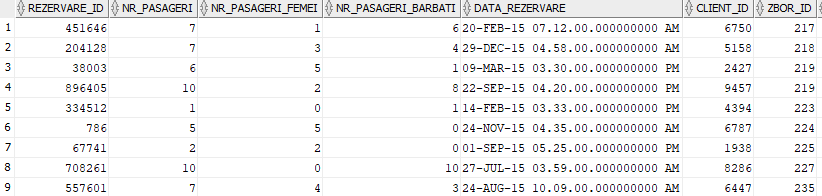
ZBOR\_NONLOWCOST

|  |
| --- |
| INSERT INTO zbor\_nonlowcost SELECT \* FROM zbor@centralizat z WHERE EXISTS (SELECT 1 FROM operator\_zbor\_nonlowcost o WHERE z.operator\_id = o.operator\_id); |



REZERVARE\_\_NONLOWCOST

|  |
| --- |
| INSERT INTO rezervare\_nonlowcost SELECT \* FROM rezervare@centralizat r WHERE EXISTS (SELECT 1 FROM zbor\_nonlowcost z WHERE r.zbor\_id = z.zbor\_id); |



PLATA\_\_NONLOWCOST

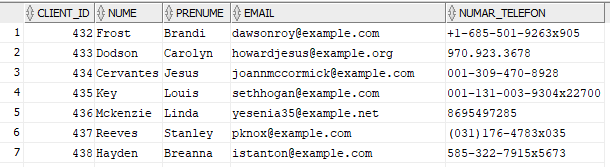
|  |
| --- |
| INSERT INTO plata\_nonlowcost SELECT p.\* FROM plata@centralizat p JOIN rezervare\_nonlowcost r ON (r.rezervare\_id = p.rezervare\_id); |



* + Fragmentarea verticală

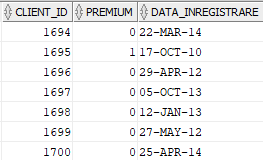
În *global\_admin* datele protejate ale clienților:

|  |
| --- |
| INSERT INTO client\_gdpr SELECT client\_id, nume, prenume, email, numar\_telefon FROM centralizat\_admin.client; |



În bazele de date locale *lowcost* și *non-lowcost*:

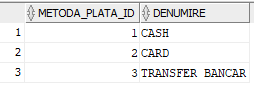
|  |
| --- |
| INSERT INTO client\_nongdpr SELECT client\_id, client\_premium, data\_inregistrare  FROM centralizat\_admin.client; |



* Popularea tabelelor replicate

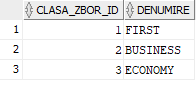
METODA\_PLATA

|  |
| --- |
| INSERT INTO metoda\_plata SELECT \* FROM metoda\_plata@centralizat; |



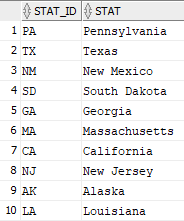
CLASA\_ZBOR

|  |
| --- |
| INSERT INTO clasa\_zbor SELECT \* FROM centralizat\_admin.clasa\_zbor; |



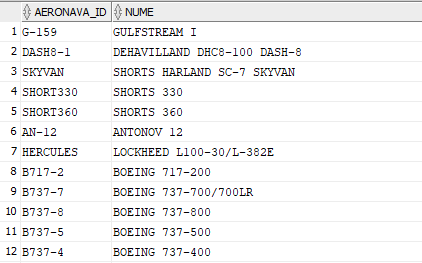
STAT

|  |
| --- |
| INSERT INTO stat SELECT \* FROM centralizat\_admin.stat; |



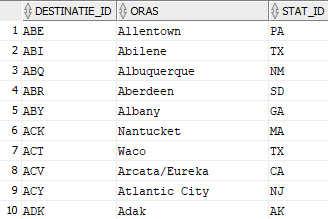
AERONAVA

|  |
| --- |
| INSERT INTO aeronava SELECT \* FROM centralizat\_admin.aeronava; |



DESTINATIE

|  |
| --- |
| INSERT INTO destinatie SELECT \* FROM centralizat\_admin.destinatie; |



1. Verificarea corectitudinii fragmentărilor realizate

Pentru a asigura corectitudinea fragmentării, trebuie analizate 3 criterii: completitudinea, reconstrucția și disjuncția. Asigurarea corectitudinii fragmentărilor poate fi realizată prin intermediul comenzilor SQL:

OPERATOR\_ZBOR

Completitudinea

|  |
| --- |
| SELECT \* FROM centralizat\_admin.operator\_zbor MINUS (SELECT \* FROM operator\_zbor\_lowcost UNION ALL SELECT \* FROM operator\_zbor\_nonlowcost@non\_lowcost); |



Reconstrucția

|  |
| --- |
| -- operator\_zbor inclus in (operator\_zbor\_lowcost U operator\_zbor\_nonlowcost) SELECT \* FROM centralizat\_admin.operator\_zbor MINUS (SELECT \* FROM operator\_zbor\_lowcost UNION ALL SELECT \* FROM operator\_zbor\_nonlowcost@non\_lowcost); |



|  |
| --- |
| -- (operator\_zbor\_lowcost U operator\_zbor\_nonlowcost) inclus in operator\_zbor (SELECT \* FROM operator\_zbor\_lowcost UNION ALL SELECT \* FROM operator\_zbor\_nonlowcost@non\_lowcost) MINUS SELECT \* FROM centralizat\_admin.operator\_zbor; |



Disjuncția

|  |
| --- |
| SELECT \* FROM operator\_zbor\_lowcost INTERSECT SELECT \* FROM operator\_zbor\_nonlowcost@non\_lowcost; |



Toate tabelele rezultate sunt goale, ceea ce indică corectitudinea fragmentării.

ZBOR

Completitudinea

|  |
| --- |
| SELECT \* FROM centralizat\_admin.zbor MINUS (SELECT \* FROM zbor\_lowcost UNION ALL SELECT \* FROM zbor\_nonlowcost@non\_lowcost); |



Reconstrucția

|  |
| --- |
| -- zbor inclus in (zbor\_lowcost U zbor\_nonlowcost) SELECT \* FROM centralizat\_admin.zbor MINUS (SELECT \* FROM zbor\_lowcost UNION ALL SELECT \* FROM zbor\_nonlowcost@non\_lowcost); |



|  |
| --- |
| -- (zbor\_lowcost U zbor\_nonlowcost) inclus in zbor (SELECT \* FROM zbor\_lowcost UNION ALL SELECT \* FROM zbor\_nonlowcost@non\_lowcost) MINUS SELECT \* FROM centralizat\_admin.zbor; |



Disjuncția

|  |
| --- |
| SELECT \* FROM zbor\_lowcost INTERSECT SELECT \* FROM zbor\_nonlowcost@non\_lowcost; |



Toate tabelele rezultate sunt goale, ceea ce indică corectitudinea fragmentării.

REZERVARE

Completitudinea

|  |
| --- |
| SELECT \* FROM centralizat\_admin.rezervare MINUS (SELECT \* FROM rezervare\_lowcost UNION ALL SELECT \* FROM rezervare\_nonlowcost@non\_lowcost); |



Reconstrucția

|  |
| --- |
| -- rezervare inclus in (rezervare\_lowcost U rezervare\_nonlowcost) SELECT \* FROM centralizat\_admin.rezervare MINUS (SELECT \* FROM rezervare\_lowcost UNION ALL SELECT \* FROM rezervare\_nonlowcost@non\_lowcost); |

|  |
| --- |
| -- (rezervare\_lowcost U rezervare\_nonlowcost) inclus in rezervare (SELECT \* FROM rezervare\_lowcost UNION ALL SELECT \* FROM rezervare\_nonlowcost@non\_lowcost) MINUS SELECT \* FROM centralizat\_admin.rezervare; |

Disjuncția:

|  |
| --- |
| --disjunctia SELECT \* FROM rezervare\_lowcost INTERSECT SELECT \* FROM rezervare\_nonlowcost@non\_lowcost; |

Toate tabelele rezultate sunt goale, ceea ce indică corectitudinea fragmentării.

PLATA

Completitudinea

|  |
| --- |
| SELECT \* FROM centralizat\_admin.plata MINUS (SELECT \* FROM plata\_lowcost UNION ALL SELECT \* FROM plata\_nonlowcost@non\_lowcost); |



Reconstrucția

|  |
| --- |
| -- plata inclus in (plata\_lowcost U plata\_nonlowcost) SELECT \* FROM centralizat\_admin.plata MINUS (SELECT \* FROM plata\_lowcost UNION ALL SELECT \* FROM plata\_nonlowcost@non\_lowcost); |

|  |
| --- |
| -- (plata\_lowcost U plata\_nonlowcost) inclus in plata (SELECT \* FROM plata\_lowcost UNION ALL SELECT \* FROM plata\_nonlowcost@non\_lowcost) MINUS SELECT \* FROM centralizat\_admin.plata; |



Disjuncția

|  |
| --- |
| SELECT \* FROM plata\_lowcost INTERSECT SELECT \* FROM plata\_nonlowcost@non\_lowcost; |



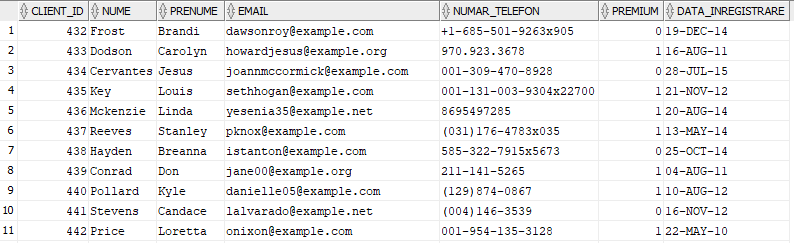
Toate tabelele rezultate sunt goale, ceea ce indică corectitudinea fragmentării.

CLIENT

Tabela client este fragmentată vertical, astfel că rezultatele în urma verificării corectitudinii, vor fi diferite:

Reconstrucția

|  |
| --- |
| SELECT gdpr.\*, nongdpr.premium, nongdpr.data\_inregistrare FROM client\_gdpr gdpr JOIN bdd\_admin.client\_nongdpr nongdpr ON (gdpr.client\_id = nongdpr.client\_id); |



În urma reconstrucției, toate coloanele din tabelele fragmentate pot fi regăsite în rezultat.

Completitudinea

|  |
| --- |
| SELECT \* FROM centralizat\_admin.client MINUS (SELECT gdpr.\*, nongdpr.premium, nongdpr.data\_inregistrare FROM client\_gdpr gdpr JOIN bdd\_admin.client\_nongdpr nongdpr ON (gdpr.client\_id = nongdpr.client\_id)); |



Tabela rezultată e goală, ceea ce indică corectitudinea completitudinii.

Disjuncția

|  |
| --- |
| SELECT column\_name FROM user\_tab\_columns WHERE table\_name = UPPER('client\_gdpr') AND column\_name <> 'CLIENT\_ID' INTERSECT SELECT column\_name FROM user\_tab\_columns WHERE table\_name = UPPER('client\_nongdpr') AND column\_name <> 'CLIENT\_ID'; |



Mulțimea coloanelor comune este vidă, deci cele două fragmente sunt disjuncte. Aceleași operații pot fi repetate si pe tabela replicată CLIENT\_NONGDPR, obținându-se aceleași rezultate.

În urma rezultatelor, poate fi concluzionat că toate fragmentările respectă toate criteriile de corectitudine.

1. Furnizarea formelor de transparență pentru întreg modelul ales

Transparența datelor permite utilizatorilor să acceseze obiectele din baza de date distantă ca și când ar fi obiecte locale. Pentru a obține acest lucru, s-au definit sinonime respectiv vizualizări în cadrul schemei fiecărui user..

Pentru utilizatorul *bdd\_admin* care se află în baza de date *orclpdb*, s-au creat sinonime pentru tabelele din schema utilizatorului *bdd\_admin* din baza de date *orclpdb\_2*.

OPERATOR\_ZBOR

|  |
| --- |
| CREATE OR REPLACE SYNONYM operator\_zbor\_nonlowcost  FOR bdd\_admin.operator\_zbor\_nonlowcost@non\_lowcost;  CREATE OR REPLACE SYNONYM operator\_zbor  FOR operator\_zbor\_lowcost; |

ZBOR

|  |
| --- |
| CREATE OR REPLACE SYNONYM zbor\_nonlowcost  FOR bdd\_admin.zbor\_nonlowcost@non\_lowcost;  CREATE OR REPLACE SYNONYM zbor  FOR zbor\_lowcost; |

REZERVARE

|  |
| --- |
| CREATE OR REPLACE SYNONYM rezervare\_nonlowcost  FOR bdd\_admin.rezervare\_nonlowcost@non\_lowcost;  CREATE OR REPLACE SYNONYM rezervare  FOR rezervare\_lowcost; |

PLATA

|  |
| --- |
| CREATE OR REPLACE SYNONYM plata\_nonlowcost  FOR bdd\_admin.plata\_nonlowcost@non\_lowcost;  CREATE OR REPLACE SYNONYM plata  FOR plata\_lowcost; |

METODA PLATA

|  |
| --- |
| CREATE OR REPLACE SYNONYM metoda\_plata\_nonlowcost  FOR bdd\_admin.metoda\_plata@non\_lowcost;  CREATE OR REPLACE SYNONYM metoda\_plata  FOR metoda\_plata\_lowcost; |

CLASA ZBOR

|  |
| --- |
| CREATE OR REPLACE SYNONYM clasa\_zbor\_nonlowcost  FOR bdd\_admin.clasa\_zbor@non\_lowcost;  CREATE OR REPLACE SYNONYM clasa\_zbor  FOR clasa\_zbor\_lowcost; |

AERONAVA

|  |
| --- |
| CREATE OR REPLACE SYNONYM aeronava\_nonlowcost  FOR bdd\_admin.aeronava@non\_lowcost;  CREATE OR REPLACE SYNONYM aeronava  FOR aeronava\_lowcost; |

DESTINATIE

|  |
| --- |
| CREATE OR REPLACE SYNONYM destinatie\_nonlowcost  FOR bdd\_admin.destinatie@non\_lowcost;  CREATE OR REPLACE SYNONYM destinatie  FOR destinatie\_lowcost; |

STAT

|  |
| --- |
| CREATE OR REPLACE SYNONYM stat\_nonlowcost  FOR bdd\_admin.stat@non\_lowcost;  CREATE OR REPLACE SYNONYM stat  FOR stat\_lowcost; |

CLIENT

|  |
| --- |
| CREATE OR REPLACE SYNONYM client\_nongdpr\_nonlowcost  FOR bdd\_admin.client\_nongdpr@non\_lowcost;  CREATE OR REPLACE SYNONYM client\_gdpr  FOR global\_admin.client\_gdpr;  CREATE OR REPLACE VIEW client AS SELECT gdpr.client\_id, gdpr.nume, gdpr.prenume, gdpr.email, gdpr.numar\_telefon, non.data\_inregistrare,non.premium  FROM client\_gdpr gdpr  JOIN client\_nongdpr non ON gdpr.client\_id = non.client\_id; |

Utilizatorul *bdd\_admin* din baza de date *orclpdb\_2* are nevoie la rândul lui nevoie de acces facil la obiectele din schema lui *bdd\_admin*din *orclpdb*.

OPERATOR ZBOR

|  |
| --- |
| CREATE OR REPLACE SYNONYM operator\_zbor\_lowcost FOR bdd\_admin.operator\_zbor\_lowcost@lowcost;  CREATE OR REPLACE SYNONYM operator\_zbor FOR operator\_zbor\_nonlowcost; |

ZBOR

|  |
| --- |
| CREATE OR REPLACE SYNONYM zbor\_lowcost FOR bdd\_admin.zbor\_nonlowcost@lowcost;  CREATE OR REPLACE SYNONYM zbor FOR zbor\_nonlowcost; |

REZERVARE

|  |
| --- |
| CREATE OR REPLACE SYNONYM rezervare\_lowcost FOR bdd\_admin.rezervare\_lowcost@lowcost;  CREATE OR REPLACE SYNONYM rezervare FOR rezervare\_nonlowcost; |

PLATA

|  |
| --- |
| CREATE OR REPLACE SYNONYM plata\_lowcost FOR bdd\_admin.plata\_lowcost@lowcost;  CREATE OR REPLACE SYNONYM plata FOR plata\_lowcost; |

METODA PLATA

|  |
| --- |
| CREATE OR REPLACE SYNONYM metoda\_plata\_lowcost FOR bdd\_admin.metoda\_plata\_lowcost@lowcost; |

CLASA ZBOR

|  |
| --- |
| CREATE OR REPLACE SYNONYM clasa\_zbor\_lowcost FOR bdd\_admin.clasa\_zbor\_lowcost@lowcost; |

AERONAVA

|  |
| --- |
| CREATE OR REPLACE SYNONYM aeronava\_lowcost FOR bdd\_admin.aeronava\_lowcost@lowcost; |

DESTINATIE

|  |
| --- |
| CREATE OR REPLACE SYNONYM destinatie\_lowcost FOR bdd\_admin.destinatie\_lowcost@lowcost; |

CLIENT

|  |
| --- |
| CREATE OR REPLACE SYNONYM client\_nongdpr\_lowcost FOR bdd\_admin.client\_nongdpr@lowcost;  CREATE OR REPLACE SYNONYM client\_gdpr FOR global\_admin.client\_gdpr@global;  CREATE OR REPLACE VIEW client AS  SELECT gdpr.client\_id, gdpr.nume, gdpr.prenume, gdpr.email, gdpr.numar\_telefon, non.data\_inregistrare,non.premium  FROM client\_gdpr gdpr  JOIN client\_nongdpr non ON gdpr.client\_id = non.client\_id; |

Pentru utilizatorul *global\_admin* din baza de date *orclpdb*, transparența este asigurată prin crearea de vizualizări către fragmentele orizontale/verticale. Prin intermediul vizualizărilor, se poate avea o privire în ansamblu asupra tuturor datelor, fără a fi clasificate în funcție de tipul operatorului de zbor. De asemenea, se definesc trigger-i de tip INSTEAD OF pe acestea pentru a asigurarea manipularea corespunzătoare a datelor.

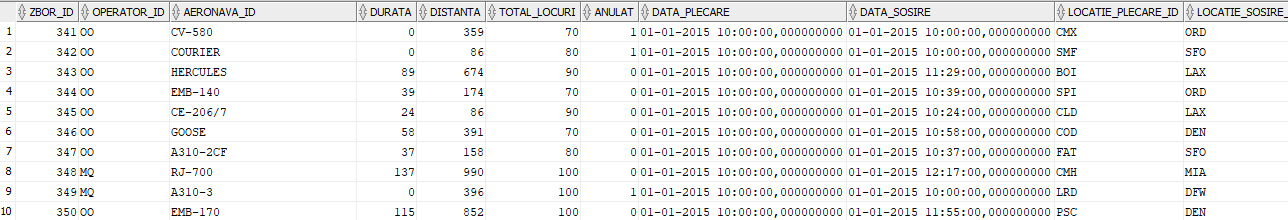
OPERATOR ZBOR

|  |
| --- |
| CREATE OR REPLACE SYNONYM operator\_zbor\_lowcost FOR bdd\_admin.operator\_zbor\_lowcost;  CREATE OR REPLACE SYNONYM operator\_zbor\_nonlowcost FOR operator\_zbor@non\_lowcost;  CREATE OR REPLACE VIEW operator\_zbor AS SELECT \* FROM operator\_zbor\_nonlowcost UNION ALL SELECT \* FROM operator\_zbor\_lowcost; |



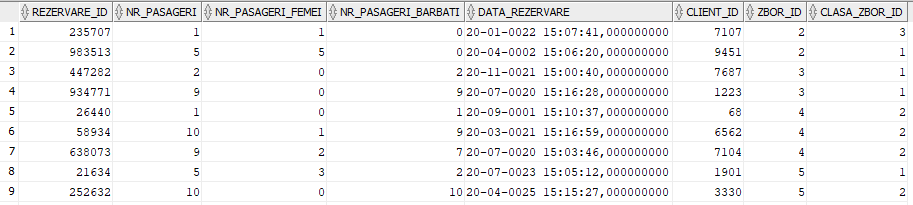
ZBOR

|  |
| --- |
| CREATE OR REPLACE SYNONYM zbor\_lowcost FOR bdd\_admin.zbor\_lowcost;  CREATE OR REPLACE SYNONYM zbor\_nonlowcost FOR zbor\_nonlowcost@non\_lowcost;  CREATE OR REPLACE VIEW zbor AS SELECT \* FROM zbor\_nonlowcost UNION ALL  SELECT \* FROM zbor\_lowcost; |



REZERVARE

|  |
| --- |
| CREATE OR REPLACE SYNONYM rezervare\_lowcost FOR bdd\_admin.rezervare\_lowcost;  CREATE OR REPLACE SYNONYM rezervare\_nonlowcost FOR rezervare\_nonlowcost@non\_lowcost;  CREATE OR REPLACE VIEW rezervare AS SELECT \* FROM rezervare\_nonlowcost UNION ALL  SELECT \* FROM rezervare\_lowcost; |



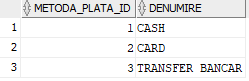
PLATA

|  |
| --- |
| CREATE OR REPLACE SYNONYM plata\_lowcost FOR bdd\_admin.plata\_lowcost;  CREATE OR REPLACE SYNONYM plata\_nonlowcost FOR plata\_nonlowcost@non\_lowcost;  CREATE OR REPLACE VIEW plata AS SELECT \* FROM plata\_nonlowcost UNION ALL  SELECT \* FROM plata\_lowcost; |



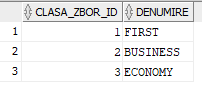
METODA PLATA

|  |
| --- |
| CREATE OR REPLACE SYNONYM metoda\_plata\_lowcost FOR bdd\_admin.metoda\_plata;  CREATE OR REPLACE SYNONYM metoda\_plata\_nonlowcost FOR metoda\_plata@non\_lowcost;  CREATE OR REPLACE VIEW metoda\_plata AS SELECT \* FROM metoda\_plata\_lowcost; |



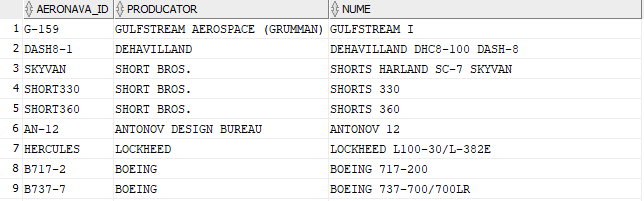
CLASA ZBOR

|  |
| --- |
| CREATE OR REPLACE SYNONYM clasa\_zbor\_lowcost FOR bdd\_admin.clasa\_zbor;  CREATE OR REPLACE SYNONYM clasa\_zbor\_nonlowcost FOR bdd\_admin.clasa\_zbor@non\_lowcost;  CREATE OR REPLACE VIEW clasa\_zbor AS SELECT \* FROM clasa\_zbor\_lowcost; |



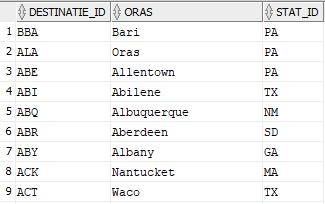
AERONAVA

|  |
| --- |
| CREATE OR REPLACE SYNONYM aeronava\_lowcost FOR bdd\_admin.aeronava;  CREATE OR REPLACE SYNONYM aeronava\_nonlowcost FOR bdd\_admin.aeronava@non\_lowcost;  CREATE OR REPLACE VIEW aeronava AS SELECT \* FROM aeronava\_lowcost; |



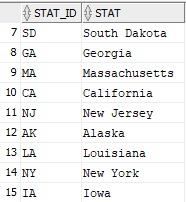
DESTINATIE

|  |
| --- |
| CREATE OR REPLACE SYNONYM destinatie\_lowcost FOR bdd\_admin.destinatie;  CREATE OR REPLACE SYNONYM destinatie\_nonlowcost FOR bdd\_admin.destinatie@non\_lowcost;  CREATE OR REPLACE VIEW destinatie AS SELECT \* FROM destinatie\_lowcost; |



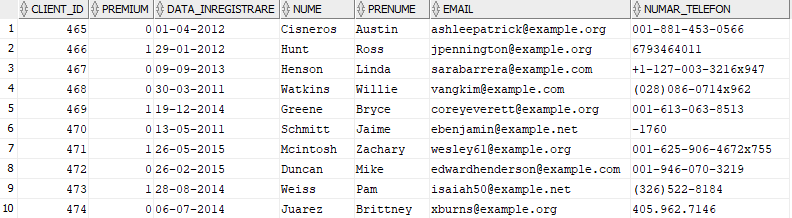
STAT

|  |
| --- |
| CREATE OR REPLACE SYNONYM stat\_lowcost FOR bdd\_admin.stat;  CREATE OR REPLACE SYNONYM stat\_nonlowcost FOR bdd\_admin.stat@non\_lowcost;  CREATE OR REPLACE VIEW stat AS SELECT \* FROM stat\_lowcost; |



CLIENT

|  |
| --- |
| CREATE OR REPLACE SYNONYM client\_nongdpr\_lowcost FOR bdd\_admin.client\_nongdpr;  CREATE OR REPLACE SYNONYM client\_nongdpr\_nonlowcost FOR bdd\_admin.client\_nongdpr@non\_lowcost;  CREATE OR REPLACE VIEW client\_nongdpr AS SELECT \* FROM client\_nongdpr\_lowcost;  CREATE OR REPLACE VIEW client AS SELECT ngdpr.client\_id, ngdpr.premium, ngdpr.data\_inregistrare, gdpr.nume, gdpr.prenume, gdpr.email, gdpr.numar\_telefon FROM client\_nongdpr ngdpr  JOIN client\_gdpr gdpr on ngdpr.client\_id = gdpr.client\_id; |



De asemenea, au fost create sinonime pentru secvențele tabelelor:

|  |
| --- |
| create or replace synonym seq\_plata  for bdd\_admin.seq\_plata;  create or replace synonym seq\_plata\_nonlowcost  for seq\_plata@non\_lowcost;  create or replace synonym seq\_client  for bdd\_admin.sec\_client\_nongdpr;  create or replace synonym seq\_rezervare  for bdd\_admin.seq\_plata;  create or replace synonym seq\_rezervare\_nonlowcost  for seq\_rezervare@non\_lowcost;  create or replace synonym seq\_zbor for bdd\_admin.seq\_zbor;  create or replace synonym seq\_zbor\_nonlowcost  for seq\_zbor@non\_lowcost;  create or replace synonym seq\_metoda\_plata for bdd\_admin.seq\_metoda\_plata;  create or replace synonym seq\_metoda\_plata\_nonlowcost  for seq\_metoda\_plata@non\_lowcost;  create or replace synonym seq\_clasa\_zbor  for bdd\_admin.seq\_clasa\_zbor;  create or replace synonym seq\_clasa\_zbor\_nonlowcost  for seq\_clasa\_zbor@non\_lowcost;  create or replace synonym seq\_client  for bdd\_admin.sec\_client\_nongdpr; |

Au fost definiți trigger-i de tipul INSTEAD OF pentru a putea gestiona actualizare datelor la nivel global. Astfel, pentru tabelele fragmentate orizontal, se verifică tipul entității ("Non low cost" sau "Low cost") iar în funcție de rezultat, se adaugă / se șterge din baza de date corespunzătoare.

OPERATOR ZBOR

|  |
| --- |
| CREATE OR REPLACE TRIGGER t\_operator\_zbor INSTEAD OF INSERT OR DELETE ON operator\_zbor FOR EACH ROW BEGIN  IF INSERTING THEN  IF :new.tip = 'Non low cost' THEN  INSERT INTO operator\_zbor\_nonlowcost (OPERATOR\_ID, NUME, TIP)   VALUES (:new.operator\_id, :new.nume, :new.tip);  ELSE  INSERT INTO operator\_zbor\_lowcost (OPERATOR\_ID, NUME, TIP)   VALUES (:new.operator\_id, :new.nume, :new.tip);  END IF;  END IF;     IF DELETING THEN  IF :old.tip = 'Non low cost' THEN  DELETE FROM operator\_zbor\_nonlowcost   WHERE operator\_id = :old.operator\_id;  ELSE   DELETE FROM operator\_zbor\_lowcost   WHERE operator\_id = :old.operator\_id;  END IF;  END IF;  END; / |

ZBOR

|  |
| --- |
| CREATE OR REPLACE TRIGGER t\_zbor INSTEAD OF INSERT OR DELETE ON zbor FOR EACH ROW DECLARE  tip VARCHAR2(15); BEGIN   IF INSERTING THEN  SELECT oz.tip INTO tip  FROM operator\_zbor oz  WHERE oz.operator\_id = :new.operator\_id;    IF tip = 'Non low cost' THEN  INSERT INTO zbor\_nonlowcost (ZBOR\_ID, OPERATOR\_ID, AERONAVA\_ID, LOCATIE\_PLECARE\_ID, LOCATIE\_SOSIRE\_ID, DATA\_PLECARE, DURATA, DISTANTA, DATA\_SOSIRE, ANULAT, TOTAL\_LOCURI)   VALUES (seq\_zbor\_nonlowcost.nextval,:new.OPERATOR\_ID, :new.AERONAVA\_ID, :new.LOCATIE\_PLECARE\_ID, :new.LOCATIE\_SOSIRE\_ID, :new.DATA\_PLECARE,:new.DURATA,:new.DISTANTA,:new.DATA\_SOSIRE,:new.ANULAT,:new.TOTAL\_LOCURI);  ELSE  INSERT INTO zbor\_lowcost (ZBOR\_ID, OPERATOR\_ID, AERONAVA\_ID, LOCATIE\_PLECARE\_ID, LOCATIE\_SOSIRE\_ID, DATA\_PLECARE, DURATA, DISTANTA, DATA\_SOSIRE, ANULAT, TOTAL\_LOCURI)   VALUES (seq\_zbor\_lowcost.nextval, :new.OPERATOR\_ID, :new.AERONAVA\_ID, :new.LOCATIE\_PLECARE\_ID, :new.LOCATIE\_SOSIRE\_ID, :new.DATA\_PLECARE,:new.DURATA,:new.DISTANTA,:new.DATA\_SOSIRE,:new.ANULAT,:new.TOTAL\_LOCURI);  END IF;  END IF;    IF DELETING THEN  SELECT oz.tip INTO tip  FROM operator\_zbor oz  WHERE oz.operator\_id = :old.operator\_id;    IF tip = 'Non low cost' THEN  DELETE FROM zbor\_nonlowcost WHERE zbor\_id = :old.zbor\_id;  ELSE  DELETE FROM zbor\_lowcost WHERE zbor\_id = :old.zbor\_id;  END IF;   END IF; END; / |

REZERVARE

|  |
| --- |
| CREATE OR REPLACE TRIGGER t\_rezervare INSTEAD OF INSERT OR DELETE ON rezervare FOR EACH ROW DECLARE  tip VARCHAR2(15); BEGIN  IF INSERTING THEN    SELECT oz.tip INTO tip  FROM operator\_zbor oz  JOIN zbor z on oz.operator\_id = z.operator\_id  WHERE :new.zbor\_id = z.zbor\_id;   IF tip = 'Non low cost' THEN   INSERT INTO rezervare\_nonlowcost (REZERVARE\_ID, NR\_PASAGERI, NR\_PASAGERI\_FEMEI, NR\_PASAGERI\_BARBATI, DATA\_REZERVARE, CLIENT\_ID, ZBOR\_ID, CLASA\_ZBOR\_ID)   VALUES (seq\_rezervare\_nonlowcost.nextval, :new.NR\_PASAGERI, :new.NR\_PASAGERI\_FEMEI, :new.NR\_PASAGERI\_BARBATI, :new.DATA\_REZERVARE,:new.CLIENT\_ID,:new.ZBOR\_ID,:new.CLASA\_ZBOR\_ID);  ELSE  INSERT INTO rezervare\_lowcost (REZERVARE\_ID, NR\_PASAGERI, NR\_PASAGERI\_FEMEI, NR\_PASAGERI\_BARBATI, DATA\_REZERVARE, CLIENT\_ID, ZBOR\_ID, CLASA\_ZBOR\_ID)   VALUES (seq\_rezervare\_lowcost.nextval, :new.NR\_PASAGERI, :new.NR\_PASAGERI\_FEMEI, :new.NR\_PASAGERI\_BARBATI, :new.DATA\_REZERVARE,:new.CLIENT\_ID,:new.ZBOR\_ID,:new.CLASA\_ZBOR\_ID);  END IF;   END IF;    IF DELETING THEN    SELECT oz.tip INTO tip  FROM operator\_zbor oz  JOIN zbor z on oz.operator\_id = z.operator\_id  WHERE :old.zbor\_id = z.zbor\_id;    IF tip = 'Non low cost' THEN  DELETE FROM rezervare\_nonlowcost WHERE rezervare\_id = :old.rezervare\_id;  ELSE   DELETE FROM rezervare\_lowcost WHERE rezervare\_id = :old.rezervare\_id;  END IF;   END IF; END; / |

PLATA

|  |
| --- |
| CREATE OR REPLACE TRIGGER t\_plata INSTEAD OF INSERT OR DELETE ON plata FOR EACH ROW DECLARE  tip VARCHAR2(15); BEGIN  IF INSERTING THEN    SELECT oz.tip INTO tip  FROM operator\_zbor oz  JOIN zbor z on oz.operator\_id = z.operator\_id  JOIN rezervare r on r.zbor\_id = z.zbor\_id  WHERE :new.rezervare\_id = r.rezervare\_id;    IF tip = 'Non low cost' THEN  INSERT INTO plata\_nonlowcost (PLATA\_ID, METODA\_PLATA\_ID, SUMA\_TOTALA, DATA\_PLATA, REZERVARE\_ID)   VALUES (seq\_plata\_nonlowcost.nextval, :new.METODA\_PLATA\_ID, :new.SUMA\_TOTALA, :new.DATA\_PLATA, :new.REZERVARE\_ID);  ELSE  INSERT INTO plata\_lowcost (PLATA\_ID, METODA\_PLATA\_ID, SUMA\_TOTALA, DATA\_PLATA, REZERVARE\_ID)   VALUES (seq\_plata\_lowcost.nextval, :new.METODA\_PLATA\_ID, :new.SUMA\_TOTALA, :new.DATA\_PLATA, :new.REZERVARE\_ID);  END IF;  END IF;    IF DELETING THEN    SELECT oz.tip INTO tip  FROM operator\_zbor oz  JOIN zbor z on oz.operator\_id = z.operator\_id  JOIN rezervare r on r.zbor\_id = z.zbor\_id  WHERE :old.rezervare\_id = r.rezervare\_id;    IF tip = 'Non low cost' THEN  DELETE FROM plata\_nonlowcost WHERE plata\_id = :old.plata\_id;  ELSE   DELETE FROM plata\_lowcost WHERE plata\_id = :old.plata\_id;  END IF;   END IF; END; / |

CLIENT

|  |
| --- |
| CREATE OR REPLACE TRIGGER t\_client\_nongdpr INSTEAD OF INSERT OR DELETE ON client\_nongdpr FOR EACH ROW DECLARE BEGIN  IF INSERTING THEN  INSERT INTO client\_nongdpr\_lowcost (CLIENT\_ID, PREMIUM, DATA\_INREGISTRARE)  VALUES(:new.CLIENT\_ID, :new.PREMIUM, :new.DATA\_INREGISTRARE);  END IF;    IF DELETING THEN  DELETE FROM client\_nongdpr\_lowcost WHERE client\_id = :old.client\_id;  END IF; END; / |

În acest trigger, se adaugă/șterge din cele 2 fragmente (orizontal si vertical).

|  |
| --- |
| CREATE OR REPLACE TRIGGER t\_client INSTEAD OF INSERT OR DELETE ON client FOR EACH ROW DECLARE nr number(1); BEGIN  IF INSERTING THEN  select count(\*) into nr  from client\_nongdpr non, client\_gdpr gdpr  where non.client\_id = gdpr.client\_id  and non.premium = :new.premium  and gdpr.email = :new.email;   if (nr<>0) then  raise\_application\_error (-20001,'Constangere de unicitate pe email si premium  incalcata. Fragmentele contin deja aceste valori');  end if;   if (:new.numar\_telefon LIKE '001-%' OR :new.numar\_telefon LIKE '+1%') then  if (:new.premium = 0) then  raise\_application\_error (-20001,'validare incalcata. Toti clientii din America din Nord  (cu prefixul 001 sau +1 la nr de telefon) trebuie sa fie clienti premium');  end if;  end if;   INSERT INTO client\_nongdpr (CLIENT\_ID, PREMIUM, DATA\_INREGISTRARE)  VALUES(:new.CLIENT\_ID, :new.PREMIUM, :new.DATA\_INREGISTRARE);    INSERT INTO client\_gdpr (CLIENT\_ID, NUME, PRENUME, EMAIL, NUMAR\_TELEFON)  VALUES(:new.CLIENT\_ID, :new.NUME, :new.PRENUME, :new.EMAIL, :new.NUMAR\_TELEFON);  END IF;    IF DELETING THEN  DELETE FROM client\_nongdpr WHERE client\_id = :old.client\_id;  DELETE FROM client\_gdpr WHERE client\_id = :old.client\_id;  END IF; END; / |

METODA PLATA

|  |
| --- |
| CREATE OR REPLACE TRIGGER t\_metoda\_plata INSTEAD OF INSERT OR DELETE ON metoda\_plata FOR EACH ROW DECLARE BEGIN  IF INSERTING THEN  INSERT INTO metoda\_plata\_lowcost (METODA\_PLATA\_ID, DENUMIRE)  VALUES(:new.METODA\_PLATA\_ID, :new.DENUMIRE);  END IF;    IF DELETING THEN  DELETE FROM metoda\_plata\_lowcost WHERE metoda\_plata\_id = :old.metoda\_plata\_id;  END IF; END; / |

CLASA ZBOR

|  |
| --- |
| CREATE OR REPLACE TRIGGER t\_clasa\_zbor INSTEAD OF INSERT OR DELETE ON clasa\_zbor FOR EACH ROW DECLARE BEGIN  IF INSERTING THEN  INSERT INTO clasa\_zbor\_lowcost(CLASA\_ZBOR\_ID, DENUMIRE)  VALUES(:new.CLASA\_ZBOR\_ID, :new.DENUMIRE);  END IF;    IF DELETING THEN  DELETE FROM clasa\_zbor\_lowcost WHERE clasa\_zbor\_id = :new.clasa\_zbor\_id;  END IF; END; / |

AERONAVA

|  |
| --- |
| CREATE OR REPLACE TRIGGER t\_aeronava INSTEAD OF INSERT OR UPDATE ON aeronava FOR EACH ROW DECLARE BEGIN  IF INSERTING THEN   INSERT INTO aeronava\_lowcost (NUME, AERONAVA\_ID)   VALUES(:new.NUME, :new.AERONAVA\_ID);  END IF;    IF DELETING THEN  DELETE FROM aeronava\_lowcost   WHERE aeronava\_id = :old.aeronava\_id;  END IF; END; / |

DESTINATIE

|  |
| --- |
| CREATE OR REPLACE TRIGGER t\_destinatie INSTEAD OF INSERT OR DELETE ON destinatie FOR EACH ROW DECLARE BEGIN  IF INSERTING THEN  INSERT INTO destinatie\_lowcost (DESTINATIE\_ID, ORAS, STAT\_ID)   VALUES(:new.DESTINATIE\_ID, :new.ORAS, :new.STAT\_ID);  END IF;    IF DELETING THEN  DELETE FROM destinatie\_lowcost WHERE destinatie\_id = :old.destinatie\_id;  END IF; END; / |

STAT

|  |
| --- |
| CREATE OR REPLACE TRIGGER t\_stat INSTEAD OF INSERT OR DELETE ON stat FOR EACH ROW DECLARE BEGIN  IF INSERTING THEN  INSERT INTO stat\_lowcost (STAT\_ID, STAT)   VALUES(:new.STAT\_ID, :new.STAT);  END IF;    IF DELETING THEN  DELETE FROM stat\_lowcost WHERE stat\_id = :old.stat\_id;  END IF; END; / |

Funcționalitatea trigger-ilor va fi demonstrată în modulul Aplicație Front-End.

1. Asigurarea sincronizării datelor pentru relațiile replicate

Tabelele care trecut prin procesul de replicare sunt METODA\_PLATA, CLASA\_ZBOR, DESTINATIE, STAT, AERONAVA și CLIENT\_NONGDPR. Acestea sunt tabelele care nu au fost fragmentate orizontal și trebuie să existe câte o copie pe ambele stații de lucru, cea corespunzătoare companiilor aeriene *low-cost,* respectiv cea a companiilor *non-low-cost.* În cazul CLIENT\_NONGDPR, acesta reprezintă un fragment vertical replicat în ambele baze de date.

În cazul tabelelor METODA\_PLATA, CLASA\_ZBOR, DESTINATIE, STAT și CLIENT\_NONGDPR, gestiunea tabelelor replicate s-a realizat prin intermediul trigger-ilor. Această metodă se potrivește mai bine în cazul de față, spre deosebire de vizualizările materializate, întrucât tabelele METODA\_PLATA, CLASA\_ZBOR, DESTINATIE și STAT au puține înregistrări, iar modificările asupra acestora sunt foarte rare, astfel că o actualizare periodică pe care o oferă vizualizările materializate nu este necesară și ar determina un consum mai mare al resurselor. În cazul tabelei CLIENT\_NONGDPR, deși are un număr considerabil mai mare de înregistrări, actualizările pot fi destul de dese și este necesară actualizarea imediată a tabelelor, pentru ca noii clienti înregistrați să aibă acces imediat în cadrul aplicației, astfel că actualizările periodice ale vizualizărilor materializate nu ar fi potrivite. Actualizarea va fi unidirecțională, datele fiind ingestate sau modificate, prin intermediul trigger-ilor, din *global* în *low-cost.* Alt set de trigger-i vor propaga modificările din tabelele *low-cost* în tabelele *non-low-cost.*

Pentru replicarea tabelei AERONAVA a fost construită o vizualizare materializată în baza de date *non-low-cost*, împreună cu tabela *log* asociată. Pentru optimizare, fost aleasă o actualizare de tip *refresh fast,* pentru a realiza actualizări mai rapide, tabelul *log* înregistrând ultimele schimbări asupra tabelei de bază. Actualizarea se realizează la fiecare minut, pentru exemplificarea propagării operațiilor, însă poate fi aleasă și o actualizare forțată.

* Replicare unidirecțională folosind trigger-i

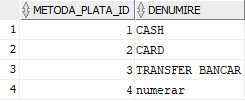
Pentru tabela METODA\_PLATA:

|  |
| --- |
| CREATE OR REPLACE TRIGGER t\_rep\_metoda\_plata AFTER INSERT OR UPDATE OR DELETE ON metoda\_plata FOR EACH ROW BEGIN  IF INSERTING THEN  INSERT INTO metoda\_plata\_nonlowcost  VALUES (:NEW.metoda\_plata\_id, :NEW.denumire);  ELSIF DELETING THEN   DELETE FROM metoda\_plata\_nonlowcost  WHERE metoda\_plata\_id = :OLD.metoda\_plata\_id;  ELSE  UPDATE metoda\_plata\_nonlowcost  SET denumire = :NEW.denumire  WHERE metoda\_plata\_id = :OLD.METODA\_PLATA\_ID;  END IF; END; / |

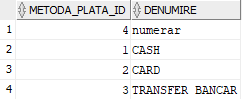
Verificare inserare:

|  |
| --- |
| INSERT INTO metoda\_plata VALUES (4, 'numerar');  SELECT \* FROM metoda\_plata; COMMIT; |

În *lowcost:*

**

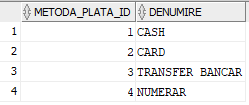
În *non-lowcost* după commit:



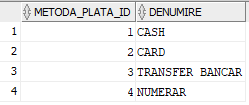
Verificare update:

|  |
| --- |
| UPDATE metoda\_plata SET denumire = 'NUMERAR' WHERE metoda\_plata\_id = 4;  SELECT \* FROM metoda\_plata; COMMIT; |

În *lowcost:*

**

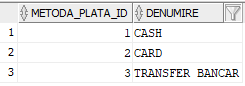
În *non-lowcost* după commit:



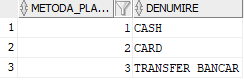
Verificare delete:

|  |
| --- |
| -- verificare stergere DELETE FROM metoda\_plata WHERE metoda\_plata\_id = 4;  SELECT \* FROM metoda\_plata; COMMIT; |

În *lowcost:*



În *non-lowcost* după commit:



Pentru tabela CLASA\_ZBOR:

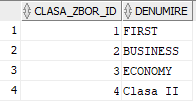
|  |
| --- |
| CREATE OR REPLACE TRIGGER t\_rep\_clasa\_zbor AFTER INSERT OR UPDATE OR DELETE ON clasa\_zbor FOR EACH ROW BEGIN  IF INSERTING THEN  INSERT INTO clasa\_zbor\_nonlowcost  VALUES (:NEW.clasa\_zbor\_id, :NEW.denumire);  ELSIF DELETING THEN   DELETE FROM clasa\_zbor\_nonlowcost  WHERE clasa\_zbor\_id = :OLD.clasa\_zbor\_id;  ELSE  UPDATE clasa\_zbor\_nonlowcost  SET denumire = :NEW.denumire  WHERE clasa\_zbor\_id = :OLD.clasa\_zbor\_id;  END IF; END; / |

Verificare trigger:

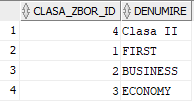
|  |
| --- |
| -- verificare inserare INSERT INTO clasa\_zbor VALUES (4, 'Clasa II');  SELECT \* FROM clasa\_zbor; COMMIT;  -- verificare update UPDATE clasa\_zbor SET denumire = 'CLASA II' WHERE clasa\_zbor\_id = 4;  SELECT \* FROM clasa\_zbor; COMMIT;  -- verificare stergere DELETE FROM clasa\_zbor WHERE clasa\_zbor\_id = 4;  SELECT \* FROM clasa\_zbor; COMMIT; |

Pentru a elimina redundanța, vom afișa doar tabelele în urma inserării:

În *lowcost:*

**

În *non-lowcost* după commit*:*



În tabela STAT:

|  |
| --- |
| CREATE OR REPLACE TRIGGER t\_rep\_stat AFTER INSERT OR UPDATE OR DELETE ON stat FOR EACH ROW BEGIN  IF INSERTING THEN  INSERT INTO stat\_nonlowcost  VALUES (:NEW.stat\_id, :NEW.stat);  ELSIF DELETING THEN   DELETE FROM stat\_nonlowcost  WHERE stat\_id = :OLD.stat\_id;  ELSE  UPDATE stat\_nonlowcost  SET stat = :NEW.stat  WHERE stat\_id = :OLD.stat\_id;  END IF; END; / |

Verificare trigger:

|  |
| --- |
| -- verificare inserare INSERT INTO stat VALUES ('NST', 'New state');  SELECT \* FROM stat; COMMIT;  -- verificare update UPDATE stat SET stat = 'NEW STATE' WHERE stat\_id = 'NST';  SELECT \* FROM stat; COMMIT;  -- verificare stergere DELETE FROM stat WHERE stat\_id = 'NST';  SELECT \* FROM stat; COMMIT; |

În baza de date *lowcost:*



În *non-lowcost* după commit:



În tabela CLIENT\_NONGDPR:

|  |
| --- |
| CREATE OR REPLACE TRIGGER t\_rep\_client\_nongdpr AFTER INSERT OR UPDATE OR DELETE ON client\_nongdpr FOR EACH ROW BEGIN  IF INSERTING THEN  INSERT INTO client\_nongdpr\_nonlowcost  VALUES (:NEW.client\_id, :NEW.premium, :NEW.data\_inregistrare);  ELSIF DELETING THEN   DELETE FROM client\_nongdpr\_nonlowcost  WHERE client\_id = :OLD.client\_id;  ELSE  UPDATE client\_nongdpr\_nonlowcost  SET premium = :NEW.premium, data\_inregistrare = :NEW.data\_inregistrare  WHERE client\_id = :OLD.client\_id;  END IF; END; / |

Verificare trigger:

|  |
| --- |
| -- verificare inserare INSERT INTO client\_nongdpr VALUES (10001, 0, sysdate);  SELECT \* FROM client\_nongdpr ORDER BY 1 DESC; COMMIT;  -- verificare update UPDATE client\_nongdpr SET premium = 1 WHERE client\_id = 10001;  SELECT \* FROM client\_nongdpr ORDER BY 1 DESC; COMMIT;  -- verificare stergere DELETE FROM client\_nongdpr WHERE client\_id = 10001;  SELECT \* FROM client\_nongdpr ORDER BY 1 DESC; COMMIT; |

În vizualizarea CLIENT:

|  |
| --- |
| CREATE OR REPLACE TRIGGER t\_client INSTEAD OF INSERT OR UPDATE OR DELETE ON client FOR EACH ROW BEGIN  IF INSERTING THEN  INSERT INTO client\_nongdpr\_nonlowcost  VALUES (:NEW.client\_id, :NEW.premium, :NEW.data\_inregistrare);  INSERT INTO client\_gdpr (CLIENT\_ID, NUME, PRENUME, EMAIL, NUMAR\_TELEFON)  VALUES (:NEW.client\_id, :NEW.nume, :NEW.prenume, :NEW.email, :NEW.numar\_telefon);  ELSIF DELETING THEN  DELETE FROM client\_nongdpr\_nonlowcost  WHERE client\_id = :OLD.client\_id;  DELETE FROM client\_gdpr  WHERE client\_id = :OLD.client\_id;  ELSE  UPDATE client\_nongdpr\_nonlowcost  SET premium = :NEW.premium, data\_inregistrare = :NEW.data\_inregistrare  WHERE client\_id = :OLD.client\_id;   UPDATE client\_gdpr  SET nume = :NEW.nume, prenume = :NEW.prenume, email = :NEW.email, numar\_telefon = :NEW.numar\_telefon  WHERE client\_id = :OLD.client\_id;  END IF; END; / |

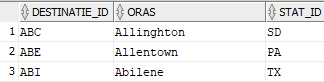
În tabela DESTINATIE:

|  |
| --- |
| CREATE OR REPLACE TRIGGER t\_rep\_destinatie AFTER INSERT OR UPDATE OR DELETE ON destinatie FOR EACH ROW BEGIN  IF INSERTING THEN  INSERT INTO destinatie\_nonlowcost  VALUES (:NEW.destinatie\_id, :NEW.oras, :NEW.stat\_id);  ELSIF DELETING THEN   DELETE FROM destinatie\_nonlowcost  WHERE destinatie\_id = :OLD.destinatie\_id;  ELSE  UPDATE destinatie\_nonlowcost  SET oras = :NEW.oras, stat\_id = :NEW.stat\_id  WHERE destinatie\_id = :OLD.destinatie\_id;  END IF; END; / |

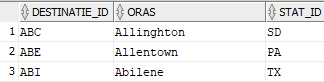
Verificare trigger:

|  |
| --- |
| -- verificare insert INSERT INTO destinatie VALUES ('ABC', 'Allinghton', 'SD');  SELECT \* FROM destinatie ORDER BY 1; COMMIT;  -- verificare update UPDATE destinatie SET oras = 'Allinghton Town' WHERE destinatie\_id = 'ABC';  SELECT \* FROM destinatie ORDER BY 1; COMMIT;  -- verificare delete DELETE FROM destinatie WHERE destinatie\_id = 'ABC'; COMMIT; |

În *lowcost:*



În *non-lowcost:*



* Replicarea unidirecțională folosind vizualizări materializate pe tabela AERONAVA

Trebuie să ne asigurăm că pe tabela AERONAVA avem definită o cheie primară.

În baza de date *lowcost,* creăm tabela *log* care va ține schimbările tabelei de bază:

|  |
| --- |
| CREATE MATERIALIZED VIEW LOG ON aeronava WITH PRIMARY KEY; |

În baza de date *non-lowcost*, creăm vizualizarea materializată pe tabela de bază AERONAVA:

|  |
| --- |
| CREATE MATERIALIZED VIEW aeronava REFRESH FAST START WITH SYSDATE NEXT SYSDATE + 1 WITH PRIMARY KEY AS SELECT \* FROM aeronava@lowcost; |

Verificăm propagarea operațiilor în vizualizarea materializată:

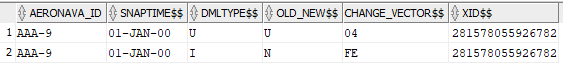
|  |
| --- |
| -- verificare insert INSERT INTO aeronava VALUES ('AAA-9', 'Boeing 181-294-142');  SELECT \* FROM aeronava ORDER BY 1;  COMMIT;  -- verificare update UPDATE aeronava  SET nume = 'Boeing 200-200-200'  WHERE aeronava\_id = 'AAA-9';  SELECT \* FROM aeronava ORDER BY 1;  COMMIT; |

În *lowcost*:



După commit în tabela de log-uri:

|  |
| --- |
| SELECT \*  FROM mlog$\_aeronava@lowcost; |



În *non-lowcost* după ce modificările au fost propagate (iar tabelul *log* e gol):

|  |
| --- |
| EXECUTE DBMS\_MVIEW.REFRESH(UPPER('aeronava'), 'F'); |

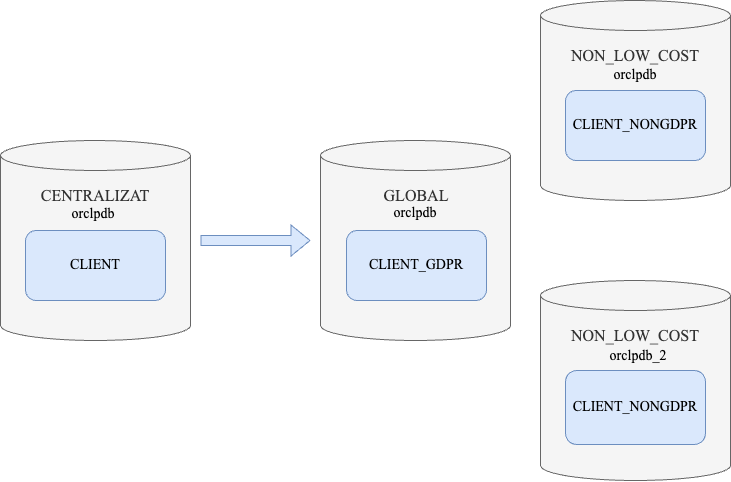


|  |
| --- |
| -- verificare delete DELETE FROM aeronava WHERE aeronava\_id = 'AAA-9';  COMMIT; |

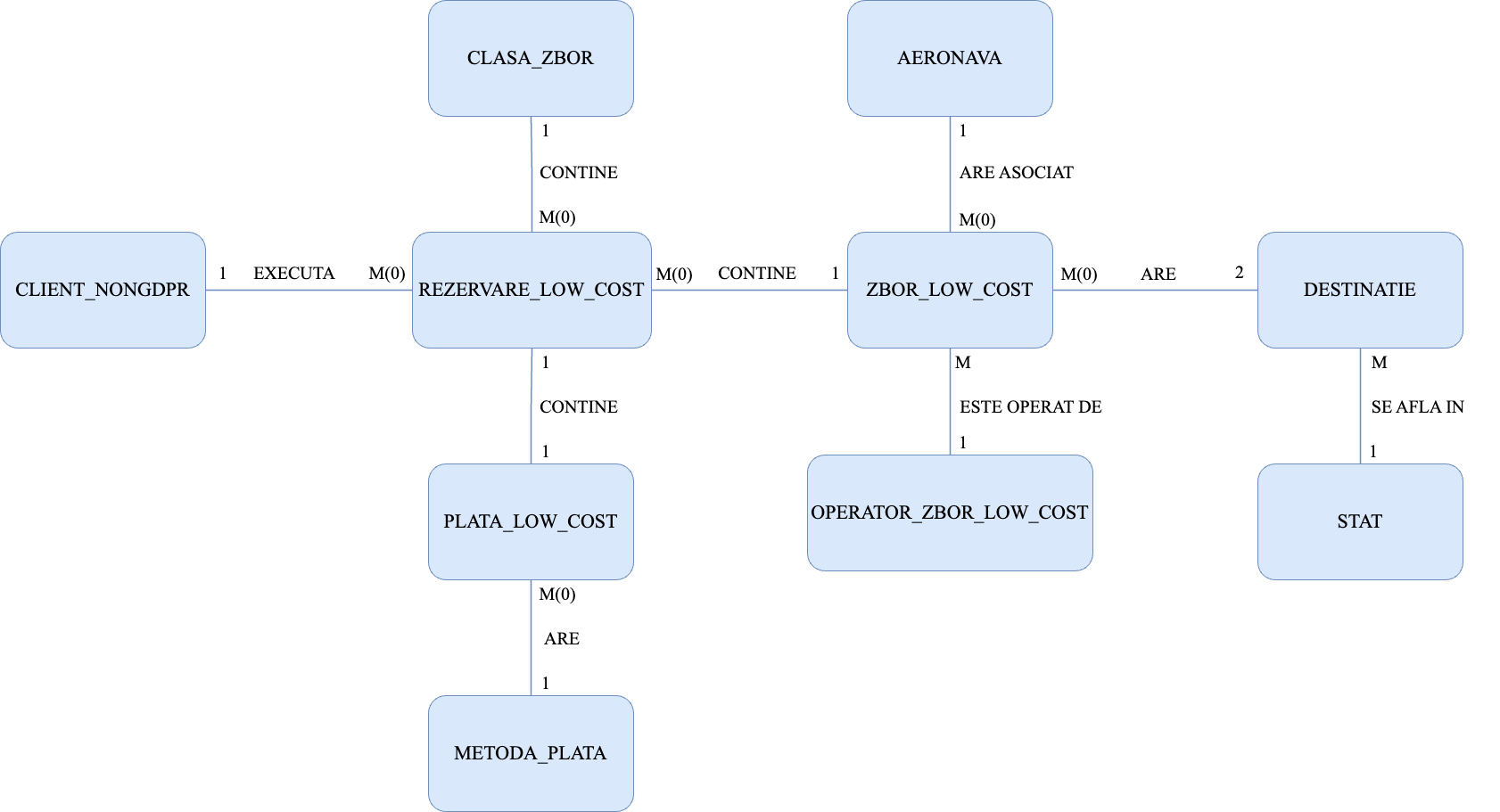


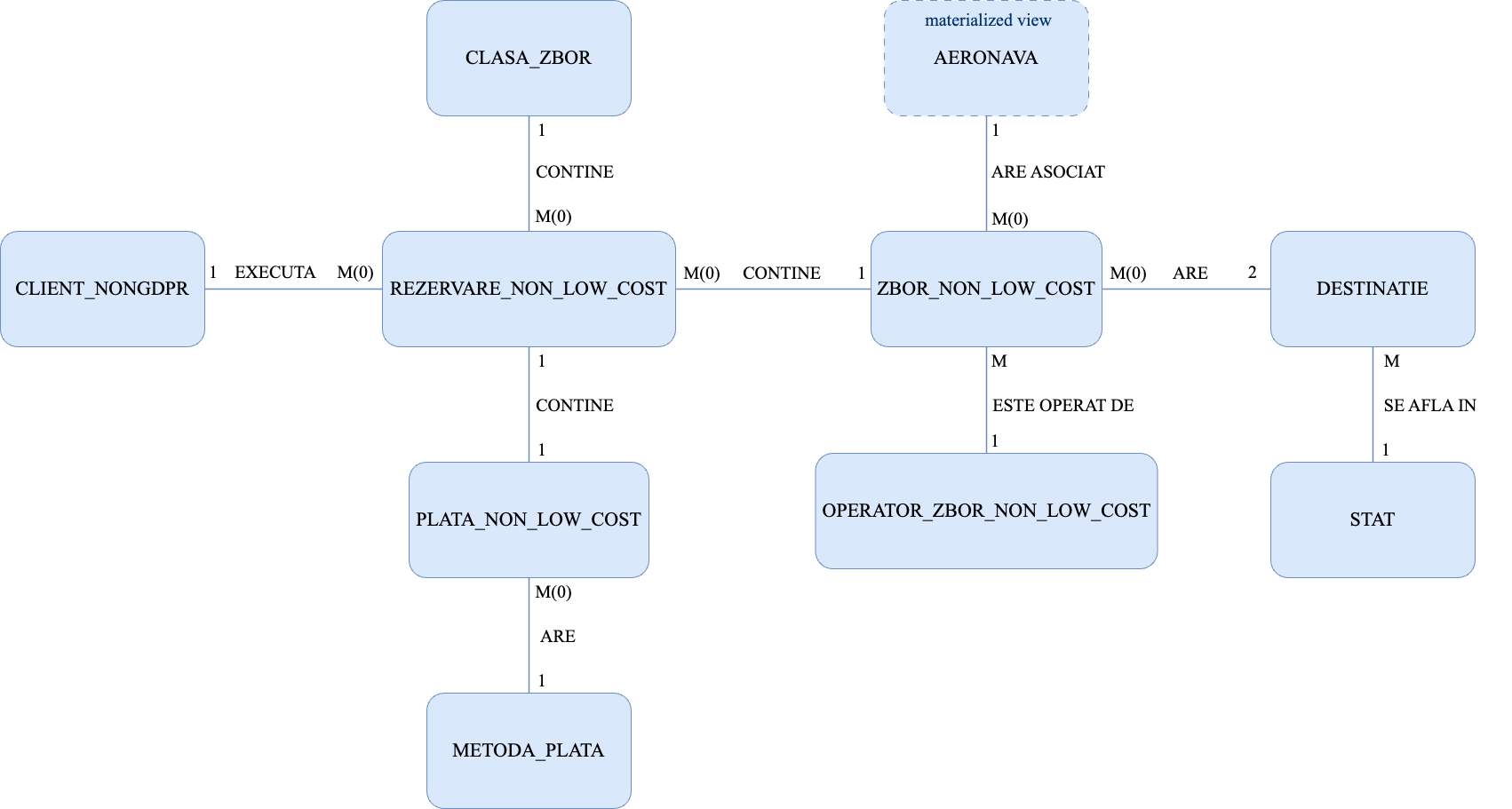
1. Rezultatele fragmentării și replicării

În urma fragmentării verticale, tabela CLIENT va avea următoarea structură:

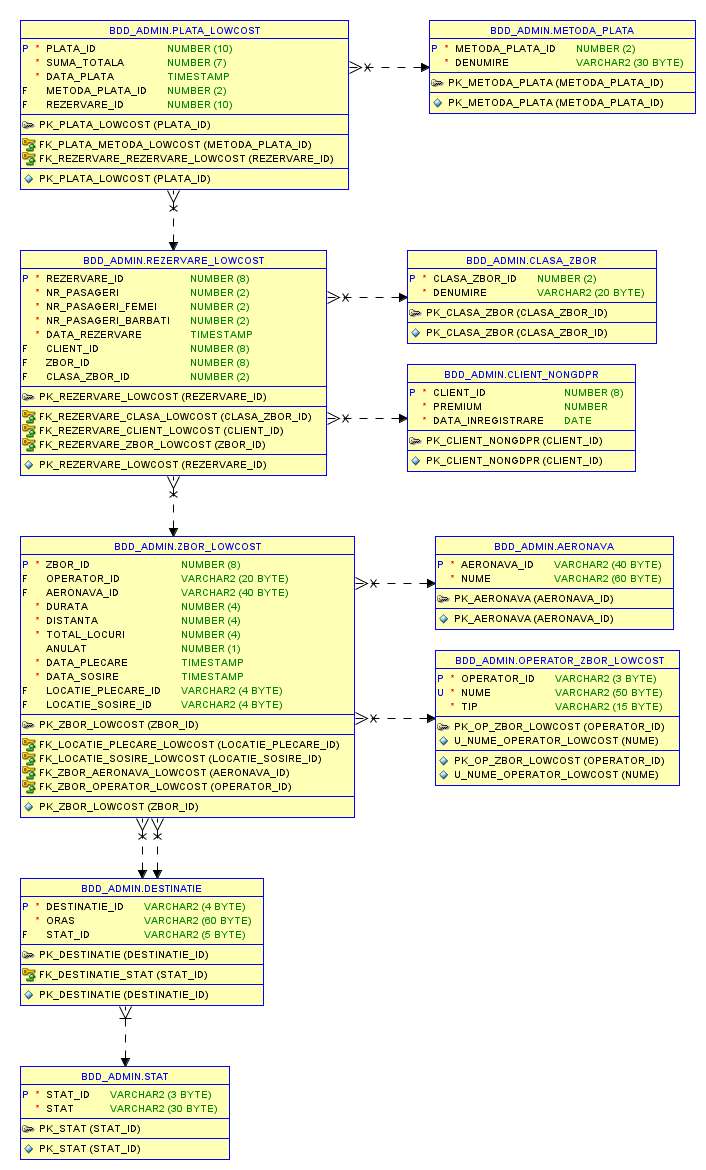
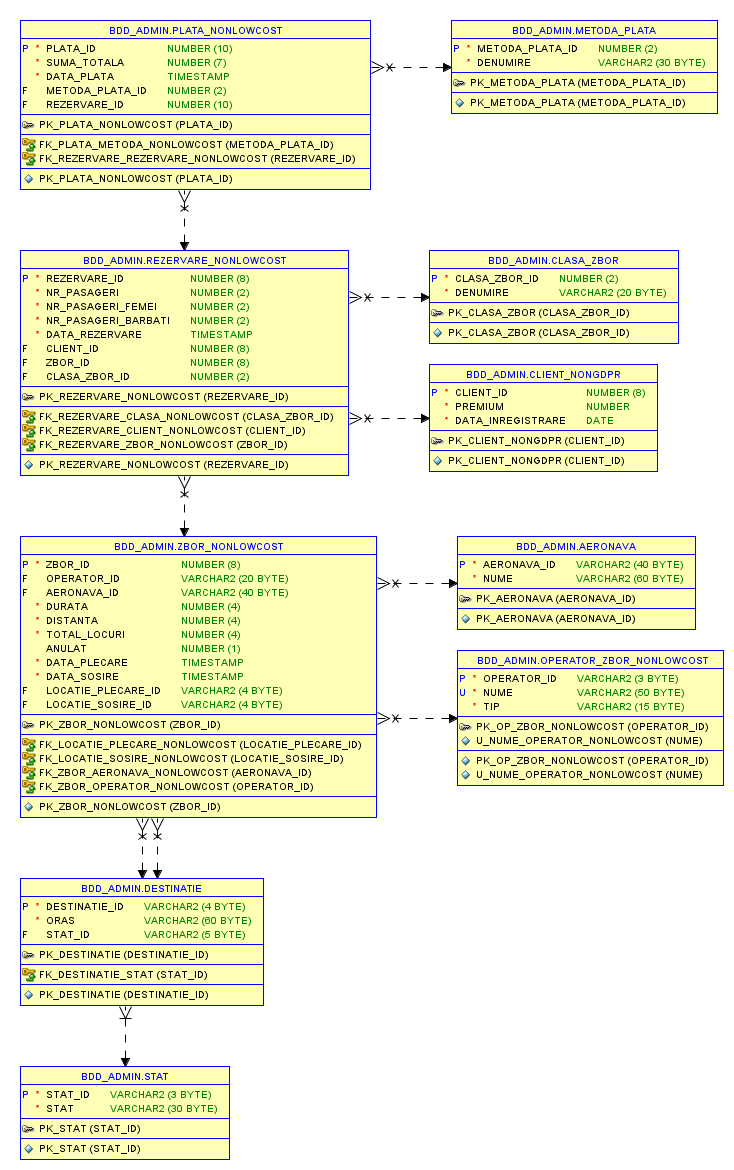


Bazele de date corespunzătoare celor două stații de lucru:

* orclpdb *(lowcost)*
* orclpdb\_2 *(lowcost)*



1. Schemele conceptuale locale corespunzătoare bazelor de date locale

* *Low cost*
* *Non low cost*

1. Asigurarea tuturor constrângerilor de integritate

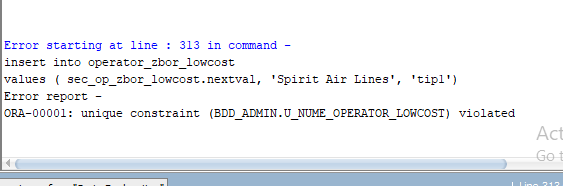
**Unicitate**

1. **Unicitatea locală**

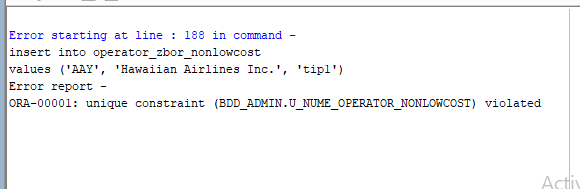
Unicitatea am asigurat-o local prin folosirea constrângerii UNIQUE în baza de date *low\_cost* pentru tabelul *operator\_zbor\_lowcost* (*nume*). La fel am procedat și pentru baza de date *non\_lowcost*, pentru tabelul *operator\_zbor\_nonlowcost* (*nume*). Am folosit-o pentru a avea valori unice ale numelui operatorului de zbor, atât local, cât și global.

|  |
| --- |
| alter table operator\_zbor\_lowcost  add constraint u\_nume\_operator\_lowcost unique (nume); |

|  |
| --- |
| insert into operator\_zbor\_lowcost values ('AYK' , 'Spirit Air Lines', 'tip1'); |



|  |
| --- |
| alter table operator\_zbor\_nonlowcost  add constraint u\_nume\_operator\_nonlowcost unique (nume);  insert into operator\_zbor\_nonlowcost values ( 'AAY', 'Hawaiian Airlines Inc.', 'tip1'); |

****

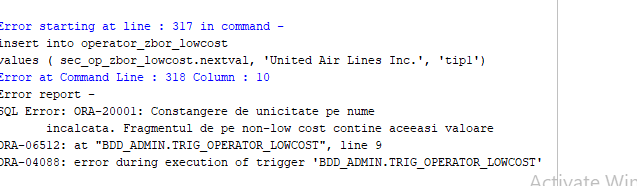
**2. Unicitatea globală pe fragmentele orizontale**

În cazul fragmentării orizontale, asigurarea constrângerii de unicitate am realizat-o folosind un trigger. Astfel, obțin valori unice ale numelui global. Trigger-ul l-am pus înainte de inserarea pe operator\_zbor\_lowcost, cautând dacă numele operatorului se regăsește și în non\_lowcost, respectiv înainte de inserarea pe operator\_zbor\_nonlowcost.

|  |
| --- |
| create or replace trigger trig\_operator\_lowcost before insert on operator\_zbor\_lowcost for each row declare nr number(1); begin select count(\*) into nr from operator\_zbor\_nonlowcost@non\_lowcost where nume = :new.nume;  if (nr<>0) then  raise\_application\_error (-20001,'Constangere de unicitate pe nume  incalcata. Fragmentul de pe non-low cost contine aceeasi valoare'); end if; end; / |

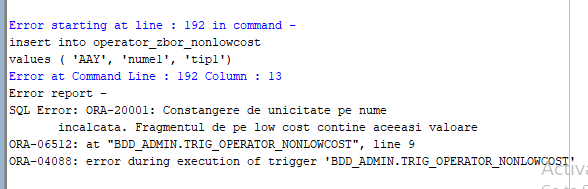
Verificăm dacă se realizează unicitatea la nivel global:

|  |
| --- |
| insert into operator\_zbor\_lowcost values ( 'AAY', 'United Air Lines Inc.', 'tip1'); |

****

La fel am procedat și pentru non\_lowcost:

|  |
| --- |
| create or replace trigger trig\_operator\_nonlowcost before insert on operator\_zbor\_nonlowcost for each row declare nr number(1); begin select count(\*) into nr from operator\_zbor\_lowcost@lowcost where nume = :new.nume;  if (nr<>0) then  raise\_application\_error (-20001,'Constangere de unicitate pe nume  incalcata. Fragmentul de pe low cost contine aceeasi valoare'); end if; end; /  insert into operator\_zbor\_nonlowcost values ( 'AAY', 'nume1', 'tip1'); |

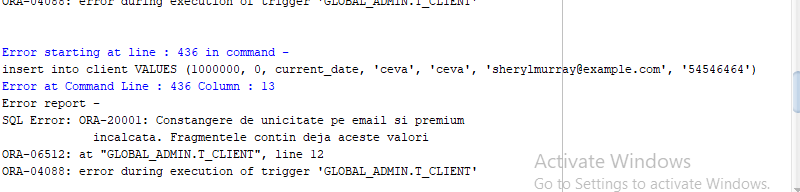
****

1. **Unicitatea globală în cazul în care trebuie să fie unică o combinație de coloane care se găsesc în fragmente verticale diferite**

În cazul fragmentării verticale, am asigurat unicitatea globală pentru combinația de coloane (email, premium), din tabelul *client\_nongdpr* de pe baza de date *lowcost* și *client\_gdpr* de pe cea globală. Prin folosirea acestei constrângeri, ne asigurăm că această combinație de valori din coloanele specificate este unică printre înregistrările din tabel, indiferent de locul unde sunt stocate. Astfel, putem preveni conflictele și inconsistențele care pot apărea. Am folosit un trigger în baza de date globală, care exista deja, și anume cel de BEFORE INSERT OR DELETE ON client. Am adăugat o verificare pentru INSERT, și anume dacă mai există combinație (email, premium) cu valorile respective în client\_gdpr, respectiv client\_nongdpr.

|  |
| --- |
| CREATE OR REPLACE TRIGGER t\_client INSTEAD OF INSERT OR DELETE ON client FOR EACH ROW DECLARE nr number(1); BEGIN  IF INSERTING THEN  select count(\*) into nr  from client\_nongdpr non, client\_gdpr gdpr  where non.client\_id = gdpr.client\_id  and non.premium = :new.premium  and gdpr.email = :new.email;   if (nr<>0) then  raise\_application\_error (-20001,'Constangere de unicitate pe email si premium  incalcata. Fragmentele contin deja aceste valori');  end if;   if (:new.numar\_telefon LIKE '001-%' OR :new.numar\_telefon LIKE '+1%') then  if (:new.premium = 0) then  raise\_application\_error (-20001,'validare incalcata. Toti clientii din America din Nord  (cu prefixul 001 sau +1 la nr de telefon) trebuie sa fie clienti premium');  end if;  end if;   INSERT INTO client\_nongdpr (CLIENT\_ID, PREMIUM, DATA\_INREGISTRARE)  VALUES(:new.CLIENT\_ID, :new.PREMIUM, :new.DATA\_INREGISTRARE);    INSERT INTO client\_gdpr (CLIENT\_ID, NUME, PRENUME, EMAIL, NUMAR\_TELEFON)  VALUES(:new.CLIENT\_ID, :new.NUME, :new.PRENUME, :new.EMAIL, :new.NUMAR\_TELEFON);  END IF;    IF DELETING THEN  DELETE FROM client\_nongdpr WHERE client\_id = :old.client\_id;  DELETE FROM client\_gdpr WHERE client\_id = :old.client\_id;  END IF; END; / |

|  |
| --- |
| insert into client VALUES (1000000, 0, current\_date, 'ceva', 'ceva', 'sherylmurray@example.com', '54546464'); |



* **Cheie primară**

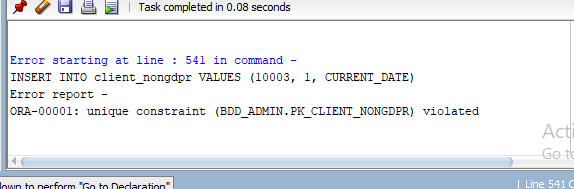
Am adăugat constrângerea de cheie primară la toate tabelele, atât în cazul fragmentării orizontale, cât și în cel al fragmentării verticale.

În cazul fragmentării orizontale, am implementat constrângerea de cheie primară prin definirea acesteia la nivelul fiecărui fragment orizontal (lowcost și nonlowcost) și prin utilizarea unui secvențe care va genera valori unice la nivel global. Secvența va genera valori pare pentru un fragment și impare pentru celălalt, pentru a asigura valori unice ale id-ului la nivel global.

În cazul fragmentării verticale, am implementat constrângerea de cheie primară prin definirea acesteia la nivelul fiecărui fragment vertical. La fel am procedat și pentru tabelele replicate.

|  |
| --- |
| alter table operator\_zbor\_lowcost  add constraint pk\_op\_zbor\_lowcost primary key (operator\_id);  alter table zbor\_lowcost  add constraint pk\_zbor\_lowcost primary key (zbor\_id);  CREATE SEQUENCE sec\_zbor\_lowcost  INCREMENT BY 2  START WITH 1048576  NOCYCLE;  alter table rezervare\_lowcost  add constraint pk\_rezervare\_lowcost primary key (rezervare\_id);  CREATE SEQUENCE sec\_rezervare\_lowcost  INCREMENT BY 2  START WITH 1040001  NOCYCLE;  alter table plata\_lowcost  add constraint pk\_plata\_lowcost primary key (plata\_id);  CREATE SEQUENCE sec\_plata\_lowcost  INCREMENT BY 2  START WITH 1040001  NOCYCLE;  alter table client\_nongdpr  add constraint pk\_client\_nongdpr primary key (client\_id);  CREATE SEQUENCE sec\_client\_nongdpr  INCREMENT BY 1  START WITH 10003  NOCYCLE;  alter table metoda\_plata  add constraint pk\_metoda\_plata primary key (metoda\_plata\_id);  CREATE SEQUENCE sec\_metoda\_plata  INCREMENT BY 1  START WITH 24  NOCYCLE;  alter table clasa\_zbor  add constraint pk\_clasa\_zbor primary key (clasa\_zbor\_id);  CREATE SEQUENCE sec\_clasa\_zbor  INCREMENT BY 1  START WITH 24  NOCYCLE;  alter table stat  add constraint pk\_stat primary key (stat\_id);  alter table aeronava  add constraint pk\_aeronava primary key (aeronava\_id);  alter table destinatie  add constraint pk\_destinatie primary key (destinatie\_id); |

|  |
| --- |
| INSERT INTO client\_nongdpr VALUES (10003, 1, CURRENT\_DATE); |



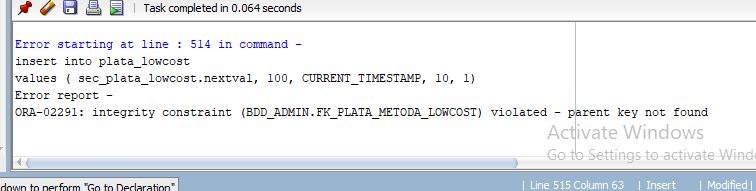
|  |
| --- |
| insert into plata\_lowcost values ( sec\_plata\_lowcost.nextval, 100, CURRENT\_TIMESTAMP, 1, 1); |

* **Cheie externă**

În cazul acestui proiect, nu avem cheie externă pentru relații stocate în baze de date diferite, ci doar la nivel local. De aceea, constrângerea de cheie externă s-a realizat local, atât pe lowcost, cât și pe nonlowcost.

|  |
| --- |
| alter table zbor\_lowcost  add constraint fk\_zbor\_operator\_lowcost FOREIGN key  (operator\_id) REFERENCES operator\_zbor\_lowcost(operator\_id)  ON DELETE CASCADE;  alter table zbor\_lowcost  add constraint fk\_zbor\_aeronava\_lowcost FOREIGN key  (aeronava\_id) REFERENCES aeronava(aeronava\_id)  ON DELETE CASCADE;  alter table zbor\_lowcost  add constraint fk\_locatie\_plecare\_lowcost FOREIGN key  (locatie\_plecare\_id) REFERENCES destinatie(destinatie\_id)  ON DELETE CASCADE;  alter table zbor\_lowcost  add constraint fk\_locatie\_sosire\_lowcost FOREIGN key  (locatie\_sosire\_id) REFERENCES destinatie(destinatie\_id)  ON DELETE CASCADE;  alter table rezervare\_lowcost  add constraint fk\_rezervare\_client\_lowcost FOREIGN key  (client\_id) REFERENCES CLIENT\_NONGDPR(client\_id)  ON DELETE CASCADE;  alter table rezervare\_lowcost  add constraint fk\_rezervare\_zbor\_lowcost FOREIGN key  (zbor\_id) REFERENCES zbor\_lowcost(zbor\_id)  ON DELETE CASCADE;  alter table rezervare\_lowcost  add constraint fk\_rezervare\_clasa\_lowcost FOREIGN key  (clasa\_zbor\_id) REFERENCES clasa\_zbor(clasa\_zbor\_id)  ON DELETE CASCADE;  alter table plata\_lowcost  add constraint fk\_plata\_metoda\_lowcost FOREIGN key  (metoda\_plata\_id) REFERENCES METODA\_PLATA(metoda\_plata\_id)  ON DELETE CASCADE;  alter table plata\_lowcost  add constraint fk\_rezervare\_rezervare\_lowcost FOREIGN key  (rezervarea\_id) REFERENCES rezervare\_lowcost(rezervare\_id)  ON DELETE CASCADE;  alter table destinatie  add constraint fk\_destinatie\_stat FOREIGN key  (stat\_id) REFERENCES stat(stat\_id); |

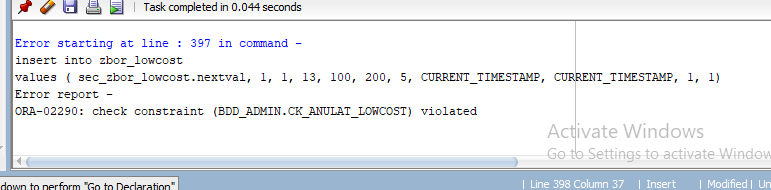
|  |
| --- |
| insert into plata\_lowcost values ( sec\_plata\_lowcost.nextval, 100, CURRENT\_TIMESTAMP, 10, 1); |



* Validare

La nivel local, unde atributele făceau parte din același fragment, constrângerea a fost adăugată local. Pentru zbor\_lowcost, avem condiția ca valoarea coloanei anulat să fie 0 sau 1. Pentru rezervare\_lowcost, nr\_pasageri, nr\_femei și nr\_bărbați trebuie să fie mai mari ca 0. Pentru client\_nongdpr, premium trebuie să fie 0 sau 1. Aceleași validări se aplică și pentru nonlowcost.

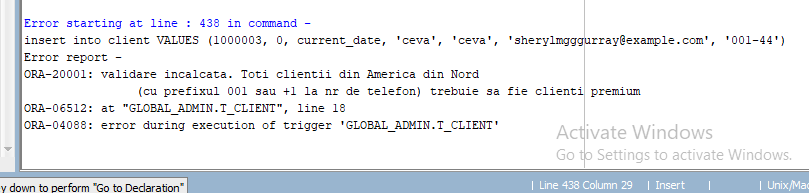
|  |
| --- |
| alter table zbor\_lowcost  add constraint ck\_anulat\_lowcost check(anulat in (0,1));  insert into zbor\_lowcost values ( sec\_zbor\_lowcost.nextval, 1, 1, 13, 100, 200, 5, CURRENT\_TIMESTAMP, CURRENT\_TIMESTAMP, 1, 1); |



Pentru relațiile stocate în baze de date diferite, am folosit un trigger pentru validare. Am dorit ca toți clienții din America din Nord (cu prefixul 001 sau +1 la nr de telefon) să fie clienți premium. În baza de date globală am adăugat într-un trigger deja existent condiția ca la inserarea în tabelul client să se verifice condiția menționată mai sus.

|  |
| --- |
| CREATE OR REPLACE TRIGGER t\_client INSTEAD OF INSERT OR DELETE ON client FOR EACH ROW DECLARE nr number(1); BEGIN  IF INSERTING THEN  select count(\*) into nr  from client\_nongdpr non, client\_gdpr gdpr  where non.client\_id = gdpr.client\_id  and non.premium = :new.premium  and gdpr.email = :new.email;   if (nr<>0) then  raise\_application\_error (-20001,'Constangere de unicitate pe email si premium  incalcata. Fragmentele contin deja aceste valori');  end if;   if (:new.numar\_telefon LIKE '001-%' OR :new.numar\_telefon LIKE '+1%') then  if (:new.premium = 0) then  raise\_application\_error (-20001,'validare incalcata. Toti clientii din America din Nord  (cu prefixul 001 sau +1 la nr de telefon) trebuie sa fie clienti premium');  end if;  end if;   INSERT INTO client\_nongdpr (CLIENT\_ID, PREMIUM, DATA\_INREGISTRARE)  VALUES(:new.CLIENT\_ID, :new.PREMIUM, :new.DATA\_INREGISTRARE);    INSERT INTO client\_gdpr (CLIENT\_ID, NUME, PRENUME, EMAIL, NUMAR\_TELEFON)  VALUES(:new.CLIENT\_ID, :new.NUME, :new.PRENUME, :new.EMAIL, :new.NUMAR\_TELEFON);  END IF;    IF DELETING THEN  DELETE FROM client\_nongdpr WHERE client\_id = :old.client\_id;  DELETE FROM client\_gdpr WHERE client\_id = :old.client\_id;  END IF; END; / |

|  |
| --- |
| insert into client VALUES (1000003, 0, current\_date, 'ceva', 'ceva', 'sherylmgggurray@example.com', '001-44'); |

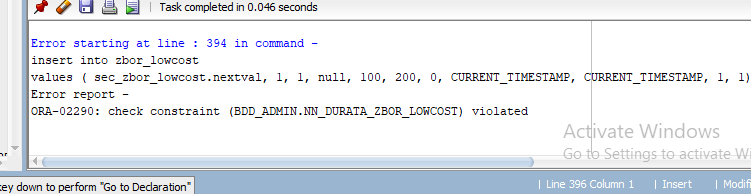


* NOT NULL

Constrângerea NOT NULL am implementat-o la nivelul tabelelor locale, pentru toate câmpurile. (atât în lowcost, nonlowcost, cât și global)

|  |
| --- |
| ALTER TABLE operator\_zbor\_lowcost  add constraint nn\_nume\_operator\_zbor\_lowcost check (nume is NOT NULL);  ALTER TABLE operator\_zbor\_lowcost  add constraint nn\_tip\_operator\_zbor\_lowcost check (tip is NOT NULL);  ALTER TABLE zbor\_lowcost  add constraint nn\_durata\_zbor\_lowcost check (durata is NOT NULL);  ALTER TABLE zbor\_lowcost  add constraint nn\_distanta\_zbor\_lowcost check (distanta is NOT NULL);  ALTER TABLE zbor\_lowcost  add constraint nn\_locuri\_zbor\_lowcost check (total\_locuri is NOT NULL);  ALTER TABLE zbor\_lowcost  add constraint nn\_data\_plecare\_zbor\_lowcost check (data\_plecare is NOT NULL);  ALTER TABLE zbor\_lowcost  add constraint nn\_data\_sosire\_zbor\_lowcost check (data\_sosire is NOT NULL);  ALTER TABLE rezervare\_lowcost  add constraint nn\_pasageri\_rezervare\_lowcost check (nr\_pasageri is NOT NULL);  ALTER TABLE rezervare\_lowcost  add constraint nn\_pasageri\_femei\_rezervare\_lowcost check (nr\_pasageri\_femei is NOT NULL);  ALTER TABLE rezervare\_lowcost  add constraint nn\_pasageri\_barbati\_rezervare\_lowcost check (nr\_pasageri\_barbati is NOT NULL);  ALTER TABLE rezervare\_lowcost  add constraint nn\_data\_rezervare\_lowcost check (data\_rezervare is NOT NULL);  ALTER TABLE plata\_lowcost  add constraint nn\_suma\_plata\_lowcost check (suma\_totala is NOT NULL);  ALTER TABLE plata\_lowcost  add constraint nn\_data\_plata\_lowcost check (data\_plata is NOT NULL); ALTER TABLE client\_nongdpr  add constraint nn\_premium\_client\_nongdpr check (premium is NOT NULL);  ALTER TABLE client\_nongdpr  add constraint nn\_data\_client\_nongdpr check (data\_inregistrare is NOT NULL);  ALTER TABLE metoda\_plata  add constraint nn\_denumire\_plata check (denumire is NOT NULL);  ALTER TABLE clasa\_zbor  add constraint nn\_denumire\_clasa\_zbor check (denumire is NOT NULL);  ALTER TABLE stat  add constraint nn\_denumire\_stat check (stat is NOT NULL);  ALTER TABLE aeronava  add constraint nn\_nume\_aeronava check (nume is NOT NULL); |

|  |
| --- |
| insert into zbor\_lowcost values ( sec\_zbor\_lowcost.nextval, 1, 1, null, 100, 200, 0, CURRENT\_TIMESTAMP, CURRENT\_TIMESTAMP, 1, 1); |



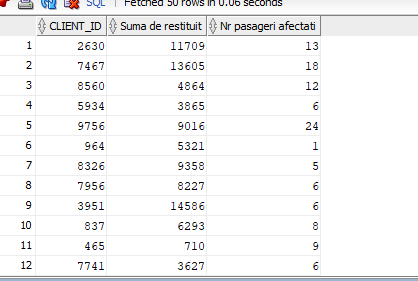
8. Optimizarea cererii SQL

*Să se afișeze suma totală pe care trebuie să o primească în luna februarie fiecare client, pentru zborurile low cost la care a achiziționat bilete, dar care au fost anulate. Clienții trebuie să fie din America de Nord (cu prefixul 001 sau +1 la nr de telefon). De asemenea, să se afișeze și numărul pasagerilor afectați.*

Această cerere va folosi date din fragmentul low cost (din fragmentarea orizontală), dar și din gdpr (din fragmentarea verticală). Ca tehnici de optimizare am folosit colectarea de statistici, creare de indecși. Statisticile sunt folosite de optimizator bazei de date pentru a estima numărul de linii care vor fi returnate de o cerere. Un avantaj al statisticilor este performanța îmbunătățită a cererii. Cu ajutorul acestora, optimizatorul poate alege cel mai eficient plan de execuție a cererii, ceea ce implică timp mai rapid de execuție al cererii. Un alt avantaj este reprezentat de utilizarea mai bună a resurselor, prin alegerea celui mai eficient plan. Un dezavantaj al statisticilor este costul de colectare și menținere. Al dezavantaj ar putea fi posibilitatea de a avea statistici imprecise, dacă nu sunt colectate și menținute corespunzător.

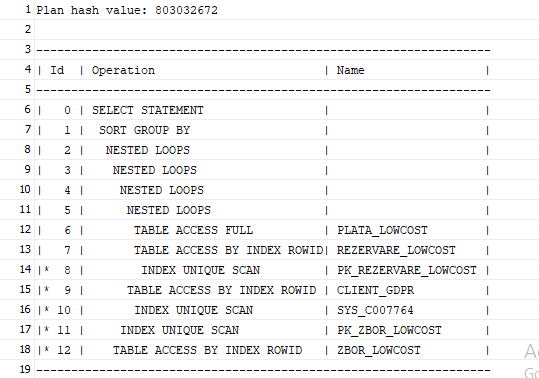
Un avantaj al indecșilor ca tehnică de optimizare este performanța mai bună a cererilor. Indecșii ajută la reducerea numărului de linii care trebuie scanate pentru a obține un răspuns la o cerere. Un dezavantaj al indecșilor este costul de creare și menținere și spațiul mare de stocare pe care îl pot necesita, mai ales pe niște tabele mari. De asemenea, pot încetini scrierile în tabel, deoarece trebuie reactualizați când sunt introduse date noi.

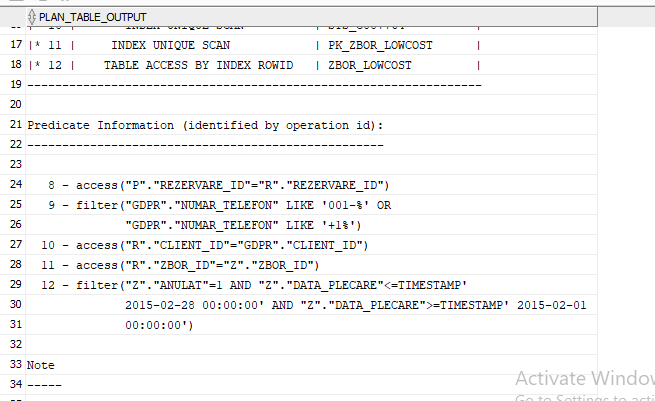
|  |
| --- |
| SELECT   gdpr.client\_id, SUM(suma\_totala) "Suma de restituit", SUM(nr\_pasageri) "Nr pasageri afectati" FROM rezervare\_lowcost r JOIN zbor\_lowcost z ON (r.zbor\_id = z.zbor\_id) JOIN plata\_lowcost p ON (p.rezervare\_id = r.rezervare\_id) JOIN global\_admin.client\_gdpr gdpr ON (r.client\_id = gdpr.client\_id) WHERE z.data\_plecare BETWEEN TO\_DATE('2015-02-01 00:00:00', 'YYYY-MM-DD hh24:mi:ss')  AND TO\_DATE('2015-02-28 00:00:00', 'YYYY-MM-DD hh24:mi:ss') AND z.anulat = 1 AND (gdpr.numar\_telefon LIKE '001-%' or gdpr.numar\_telefon LIKE '+1%') group BY gdpr.client\_id; |



* Fără colectare de statistici:

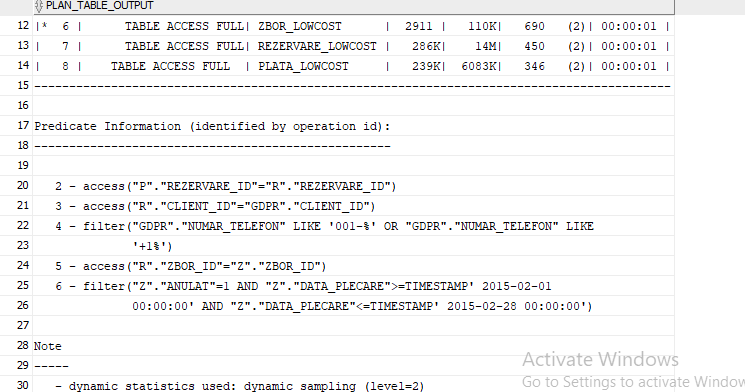
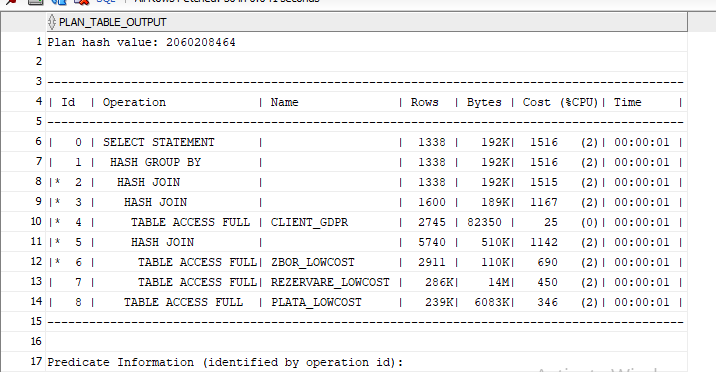
|  |
| --- |
| ALTER SESSION SET OPTIMIZER\_MODE = rule;  --rule EXPLAIN PLAN  SET STATEMENT\_ID = 'st1\_lowcost'  FOR  SELECT   gdpr.client\_id, SUM(suma\_totala) "Suma de restituit", SUM(nr\_pasageri) "Nr pasageri afectati" FROM rezervare\_lowcost r JOIN zbor\_lowcost z ON (r.zbor\_id = z.zbor\_id) JOIN plata\_lowcost p ON (p.rezervare\_id = r.rezervare\_id) JOIN global\_admin.client\_gdpr gdpr ON (r.client\_id = gdpr.client\_id) WHERE z.data\_plecare BETWEEN TO\_DATE('2015-02-01 00:00:00', 'YYYY-MM-DD hh24:mi:ss')  AND TO\_DATE('2015-02-28 00:00:00', 'YYYY-MM-DD hh24:mi:ss') AND z.anulat = 1 AND (gdpr.numar\_telefon LIKE '001-%' or gdpr.numar\_telefon LIKE '+1%') group BY gdpr.client\_id;  SELECT plan\_table\_output FROM  table(dbms\_xplan.display('PLAN\_TABLE', 'st1\_lowcost','SERIAL')); |





Nu am colectat statistici, metoda de optimizare este RULE, algoritmul de procesare a operației de join este NESTED LOOPS, iar pentru accesarea datelor s-a folosit ACCESS BY INDEX ROWID și ACCESS FULL.

|  |
| --- |
| --all rows ALTER SESSION SET OPTIMIZER\_MODE = all\_rows;  EXPLAIN PLAN  SET STATEMENT\_ID = 'st2\_lowcost'  FOR  SELECT   gdpr.client\_id, SUM(suma\_totala) "Suma de restituit", SUM(nr\_pasageri) "Nr pasageri afectati" FROM rezervare\_lowcost r JOIN zbor\_lowcost z ON (r.zbor\_id = z.zbor\_id) JOIN plata\_lowcost p ON (p.rezervare\_id = r.rezervare\_id) JOIN global\_admin.client\_gdpr gdpr ON (r.client\_id = gdpr.client\_id) WHERE z.data\_plecare BETWEEN TO\_DATE('2015-02-01 00:00:00', 'YYYY-MM-DD hh24:mi:ss')  AND TO\_DATE('2015-02-28 00:00:00', 'YYYY-MM-DD hh24:mi:ss') AND z.anulat = 1 AND (gdpr.numar\_telefon LIKE '001-%' or gdpr.numar\_telefon LIKE '+1%') group BY gdpr.client\_id;  SELECT plan\_table\_output FROM  table(dbms\_xplan.display('PLAN\_TABLE', 'st2\_lowcost')); |



Nu am colectat statistici, metoda de optimizare este ALL ROWS, algoritmul de procesare a operației de join este HASH JOIN, iar pentru accesarea datelor s-a folosit ACCESS FULL.

|  |
| --- |
| EXPLAIN PLAN  SET STATEMENT\_ID = 'st3\_lowcost'  FOR  SELECT /\*+ CHOOSE \*/  gdpr.client\_id, SUM(suma\_totala) "Suma de restituit", SUM(nr\_pasageri) "Nr pasageri afectati" FROM rezervare\_lowcost r JOIN zbor\_lowcost z ON (r.zbor\_id = z.zbor\_id) JOIN plata\_lowcost p ON (p.rezervare\_id = r.rezervare\_id) JOIN global\_admin.client\_gdpr gdpr ON (r.client\_id = gdpr.client\_id) WHERE z.data\_plecare BETWEEN TO\_DATE('2015-02-01 00:00:00', 'YYYY-MM-DD hh24:mi:ss')  AND TO\_DATE('2015-02-28 00:00:00', 'YYYY-MM-DD hh24:mi:ss') AND z.anulat = 1 AND (gdpr.numar\_telefon LIKE '001-%' or gdpr.numar\_telefon LIKE '+1%') group BY gdpr.client\_id;  SELECT plan\_table\_output FROM  table(dbms\_xplan.display('PLAN\_TABLE', 'st3\_lowcost')); |

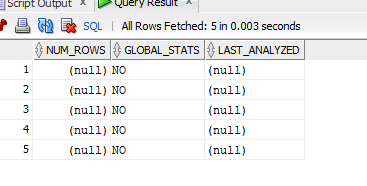
Nu am colectat statistici, metoda de optimizare este CHOOSE, transmisă printr-un hint, iar restul caracteristicilor sunt la fel ca cele de la RULE.

* Cu colectare de statistici:

--colectam statistici

|  |
| --- |
| select NUM\_ROWS, GLOBAL\_STATS, LAST\_ANALYZED from USER\_TABLES where table\_name in (upper('zbor\_lowcost'),upper('rezervare\_lowcost'), upper('plata\_lowcost'),  upper('clasa\_zbor'), upper('client\_nongdpr')); |

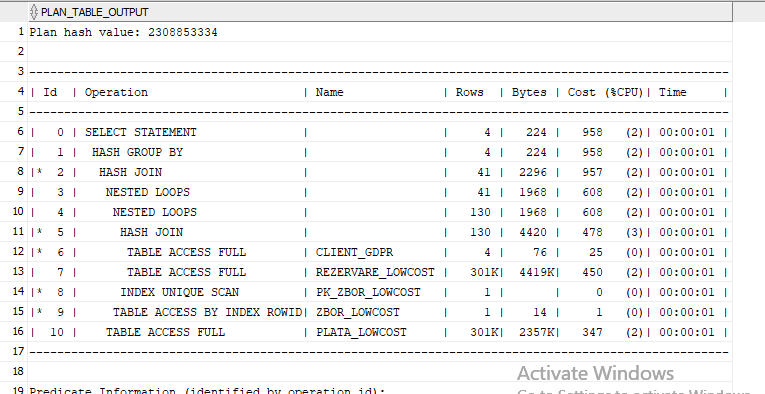
Am verificat dacă pentru tabelele accesate în cerere am colectat statistici până în acel moment.



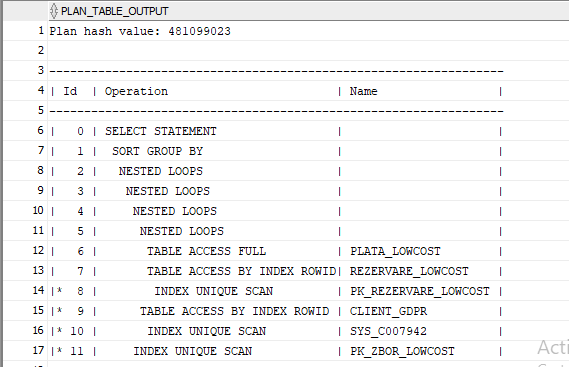
Am colectat statistici pentru tabelele accesate în cerere.

|  |
| --- |
| ANALYZE TABLE zbor\_lowcost COMPUTE STATISTICS; ANALYZE TABLE rezervare\_lowcost COMPUTE STATISTICS; ANALYZE TABLE plata\_lowcost COMPUTE STATISTICS; ANALYZE TABLE clasa\_zbor COMPUTE STATISTICS; ANALYZE TABLE client\_nongdpr COMPUTE STATISTICS; ANALYZE TABLE global\_admin.client\_gdpr COMPUTE STATISTICS; |

|  |
| --- |
| ALTER SESSION SET OPTIMIZER\_MODE = choose;  --choose EXPLAIN PLAN  SET STATEMENT\_ID = 'st4\_lowcost'  FOR SELECT   gdpr.client\_id, SUM(suma\_totala) "Suma de restituit", SUM(nr\_pasageri) "Nr pasageri afectati" FROM rezervare\_lowcost r JOIN zbor\_lowcost z ON (r.zbor\_id = z.zbor\_id) JOIN plata\_lowcost p ON (p.rezervare\_id = r.rezervare\_id) JOIN global\_admin.client\_gdpr gdpr ON (r.client\_id = gdpr.client\_id) WHERE z.data\_plecare BETWEEN TO\_DATE('2015-02-01 00:00:00', 'YYYY-MM-DD hh24:mi:ss')  AND TO\_DATE('2015-02-28 00:00:00', 'YYYY-MM-DD hh24:mi:ss') AND z.anulat = 1 AND (gdpr.numar\_telefon LIKE '001-%' or gdpr.numar\_telefon LIKE '+1%') group BY gdpr.client\_id;  SELECT plan\_table\_output FROM  table(dbms\_xplan.display('PLAN\_TABLE', 'st4\_lowcost')); |

 Costul de optimizare este 958, metoda de optimizare este choose, optimizatorul este bazat pe cost, algoritmii de procesare a operației de join sunt HASH JOIN și NESTED LOOPS, iar pentru accesarea datelor s-a folosit ACCESS FULL și ACCESS BY INDEX ROWID.

|  |
| --- |
| ALTER SESSION SET OPTIMIZER\_MODE = rule;  --rule EXPLAIN PLAN  SET STATEMENT\_ID = 'st5\_lowcost'  FOR  SELECT  gdpr.client\_id, SUM(suma\_totala) "Suma de restituit", SUM(nr\_pasageri) "Nr pasageri afectati" FROM rezervare\_lowcost r JOIN zbor\_lowcost z ON (r.zbor\_id = z.zbor\_id) JOIN plata\_lowcost p ON (p.rezervare\_id = r.rezervare\_id) JOIN global\_admin.client\_gdpr gdpr ON (r.client\_id = gdpr.client\_id) WHERE z.data\_plecare BETWEEN TO\_DATE('2015-02-01 00:00:00', 'YYYY-MM-DD hh24:mi:ss')  AND TO\_DATE('2015-02-28 00:00:00', 'YYYY-MM-DD hh24:mi:ss') AND z.anulat = 1 AND (gdpr.numar\_telefon LIKE '001-%' or gdpr.numar\_telefon LIKE '+1%') group BY gdpr.client\_id;  SELECT plan\_table\_output FROM  table(dbms\_xplan.display('PLAN\_TABLE', 'st5\_lowcost')); |



Metoda de optimizare este RULE, algoritmul de procesare a operației de join este NESTED LOOPS, iar pentru accesarea datelor s-a folosit ACCESS BY INDEX ROWID și ACCESS FULL.

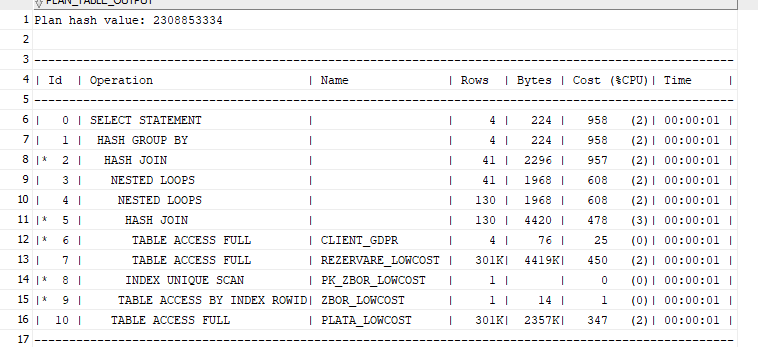
* Optimizare cu indecși

Cum tabelele *Rezervare* și *Zbor* au un număr mare de înregistrări în tabele, se va defini un index de tip *bitmap join*  între cele 2 pe coloana *anulat*. Index-ul va fi de tip bitmap întrucât există un număr limitat de valori posibile pentru anulat. Colectez și statistici pentru acest index.

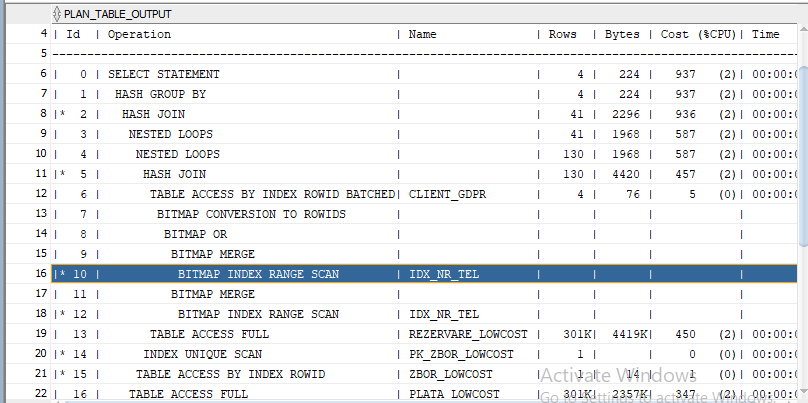
|  |
| --- |
| CREATE BITMAP INDEX idx\_zbor\_anulat ON zbor\_lowcost(anulat); ANALYZE INDEX idx\_zbor\_anulat compute statistics; |

|  |
| --- |
| ALTER SESSION SET OPTIMIZER\_MODE = choose;  --choose EXPLAIN PLAN  SET STATEMENT\_ID = 'st8\_lowcost'  FOR  SELECT   gdpr.client\_id, SUM(suma\_totala) "Suma de restituit", SUM(nr\_pasageri) "Nr pasageri afectati" FROM rezervare\_lowcost r JOIN zbor\_lowcost z ON (r.zbor\_id = z.zbor\_id) JOIN plata\_lowcost p ON (p.rezervare\_id = r.rezervare\_id) JOIN global\_admin.client\_gdpr gdpr ON (r.client\_id = gdpr.client\_id) WHERE z.data\_plecare BETWEEN TO\_DATE('2015-02-01 00:00:00', 'YYYY-MM-DD hh24:mi:ss')  AND TO\_DATE('2015-02-28 00:00:00', 'YYYY-MM-DD hh24:mi:ss') AND z.anulat = 1 AND (gdpr.numar\_telefon LIKE '001-%' or gdpr.numar\_telefon LIKE '+1%') group BY gdpr.client\_id;  SELECT plan\_table\_output FROM  table(dbms\_xplan.display('PLAN\_TABLE', 'st8\_lowcost')); |

Costul de optimizare este 958, metoda de optimizare este choose, optimizatorul este bazat pe cost, algoritmii de procesare a operației de join sunt HASH JOIN și NESTED LOOPS, iar pentru accesarea datelor s-a folosit ACCESS FULL și ACCESS BY INDEX ROWID. Observăm că costul a rămas la fel, deci indexul nu a adus o îmbunătățire.



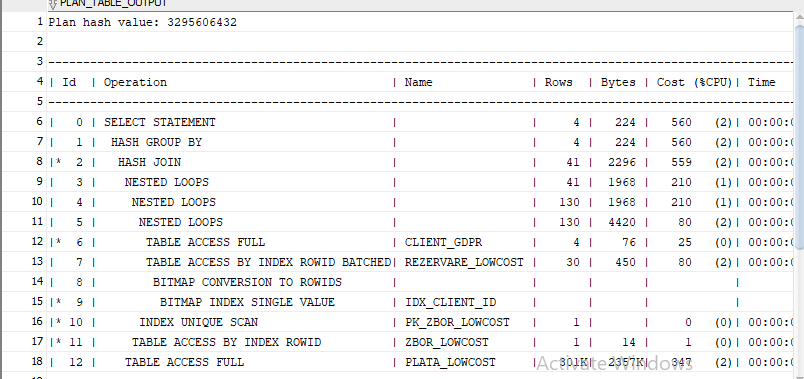
|  |
| --- |
| CREATE INDEX idx\_nr\_tel ON global\_admin.client\_gdpr(numar\_telefon); ANALYZE INDEX idx\_nr\_tel compute statistics; |



Cu un index pe nr de telefon al clientului, din tabelul client\_gdpr, costul de optimizare este 937, metoda de optimizare este choose, optimizatorul este bazat pe cost, algoritmii de procesare a operației de join sunt HASH JOIN și NESTED LOOPS, iar pentru accesarea datelor s-a folosit ACCESS FULL și ACCESS BY INDEX ROWID, ACCESS BY INDEX ROWID BATCHED și BITMAP INDEX RANGE SCAN. Observăm că costul s-a îmbunătățit, dar nu semnificativ.

|  |
| --- |
| CREATE BITMAP INDEX idx\_client\_id ON rezervare\_lowcost(client\_id); ANALYZE INDEX idx\_client\_id compute statistics; |

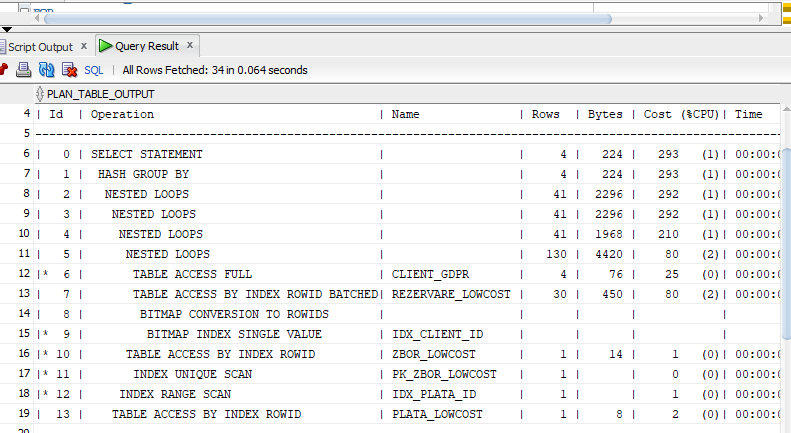
Am creat un index pe cheia străină din rezervare\_lowcost, și anume client\_id, pentru a optimiza cererea. Astfel, când se face join-ul între rezervare\_lowcost și client\_gdpr, este mai rapid și duce la o execuție mai rapidă a cererii.



Costul de optimizare se micșorează considerabil și devine 560, metoda de optimizare este choose, optimizatorul este bazat pe cost, algoritmii de procesare a operației de join sunt HASH JOIN și NESTED LOOPS, iar pentru accesarea datelor s-a folosit ACCESS FULL și ACCESS BY INDEX ROWID, ACCESS BY INDEX ROWID BATCHED și BITMAP INDEX RANGE SCAN.

|  |
| --- |
| CREATE INDEX idx\_plata\_id ON plata\_lowcost(rezervare\_id); ANALYZE INDEX idx\_plata\_id delete statistics; |

Nu în ultimul rând, am creat un index pe cheia străină din plata\_lowcost, și anume rezervare\_id, pentru a optimiza cererea. Astfel, când se face join-ul între rezervare\_lowcost și plata\_lowcost, este mai rapid și duce la o execuție mai rapidă a cererii.



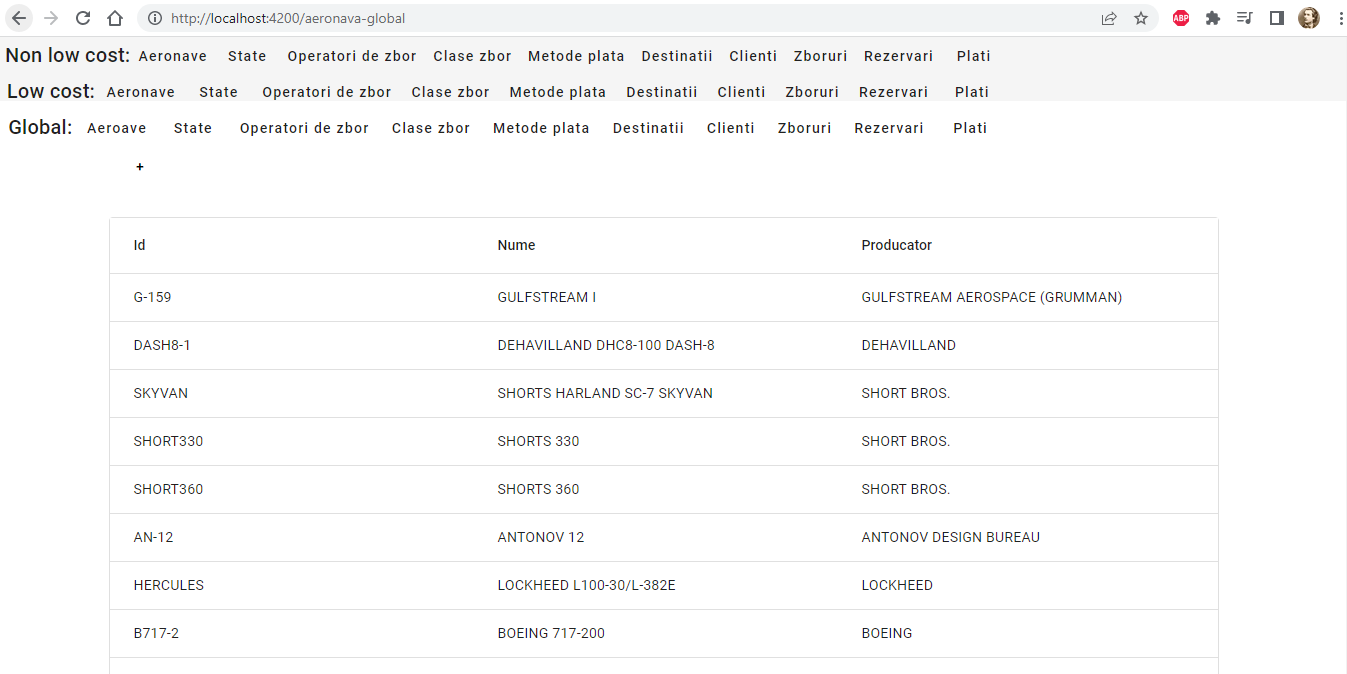
Costul de optimizare se micșorează considerabil și devine 293, metoda de optimizare este choose, optimizatorul este bazat pe cost, algoritmul de procesare a operației de join este NESTED LOOPS, iar pentru accesarea datelor s-a folosit ACCESS FULL și ACCESS BY INDEX ROWID, ACCESS BY INDEX ROWID BATCHED și BITMAP INDEX RANGE SCAN.

Această valoare a costului este cea mai bună de până acum, ceea ce înseamnă că optimizarea cu indexul pe cheia străină din plata\_lowcost, și anume rezervare\_id, este cea mai eficientă.

9. Modulul aplicație

1. Introducerea și gestionarea informațiilor la nivelul bazelor de date locale

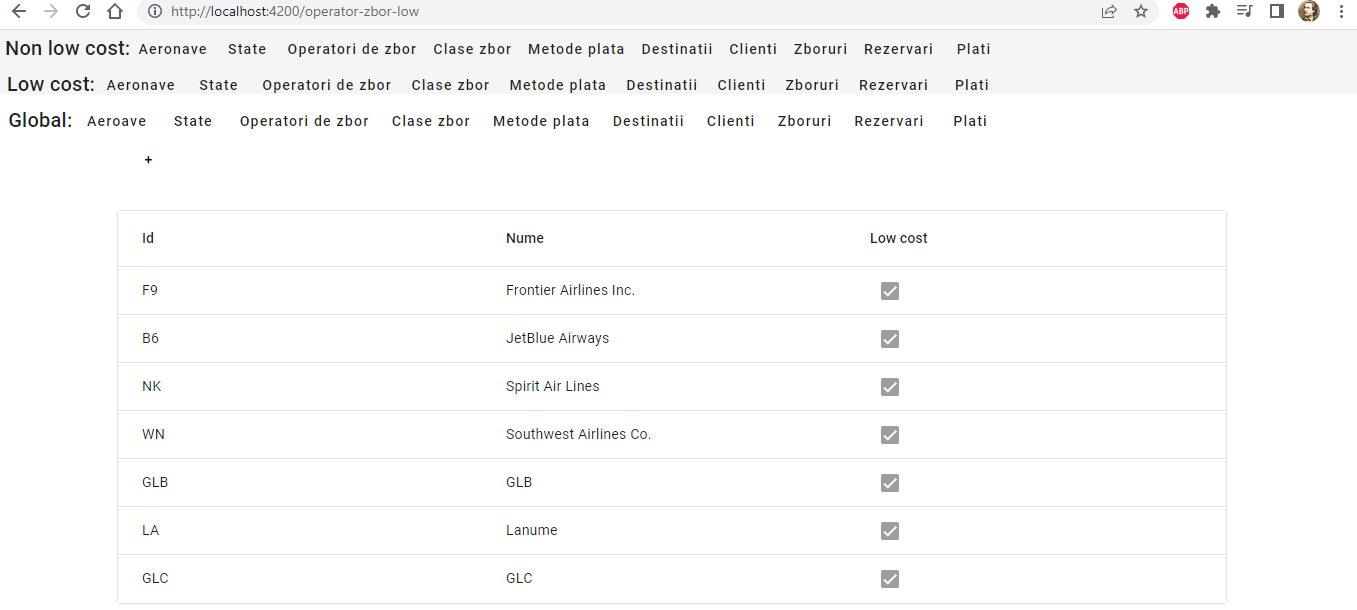
Pentru modulul “Aplicație”, a fost realizată o aplicație Web folosind Typescript/Angular pentru partea de frontend, respectiv Java/Spring Boot pentru backend. Prin intermediul aplicației, utilizatorul are dreptul să vizualizeze datele din cele 2 servere de baze de date diferite: *orclpdb* (lowcost, global), respectiv *orclpdb\_2* (nonlowcost). De asemenea, el are posibilitatea să insereze date în toate bazele de date, urmând ca acestea să fie propagate. Pagina principală a aplicației conține un widget de tip navbar prin care utilizatorul poate accesa datele în funcție de baza de date aleasă:



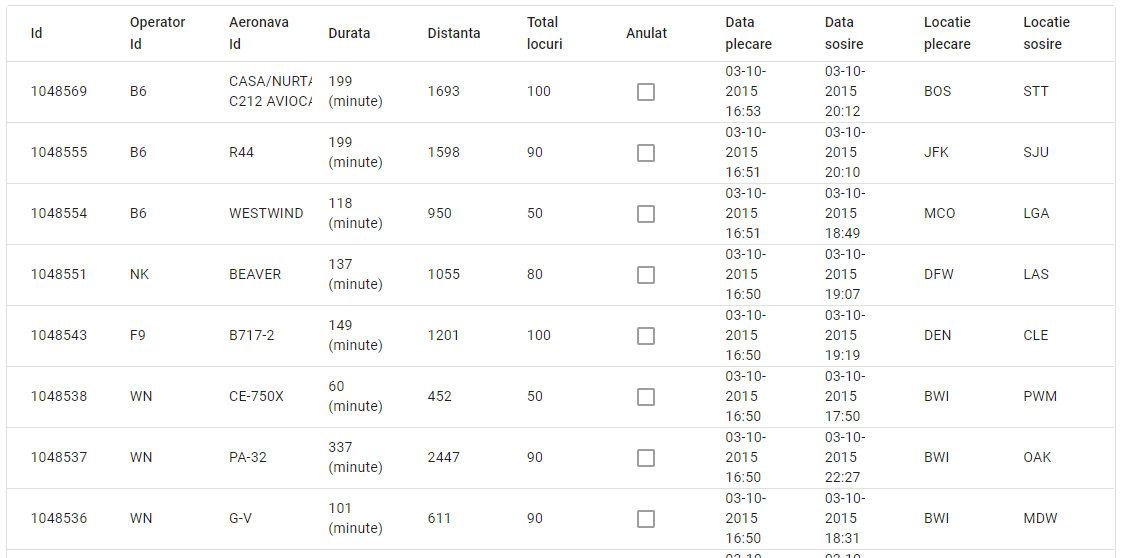
Acesta poate naviga și vizualiza date despre oricare din categorii: *Aeronave*, *State*, *Operatori de Zbor*, *Clase de Zbor*, *Metode de plată, Destinații*, *Clienți*, *Zboruri*, *Rezervări*, *Plăți*, atât la nivel local (*Low cost* / *Non low cost*), cât și global

Exemplificăm vizualizarea informațiilor la nivel local pentru datele *Low cost*:

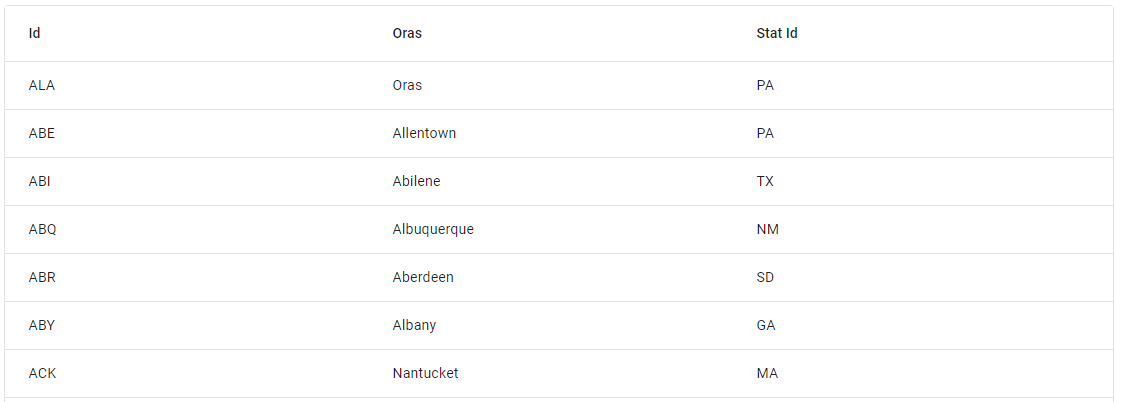
* Operator zbor (tabel fragmentat orizontal)



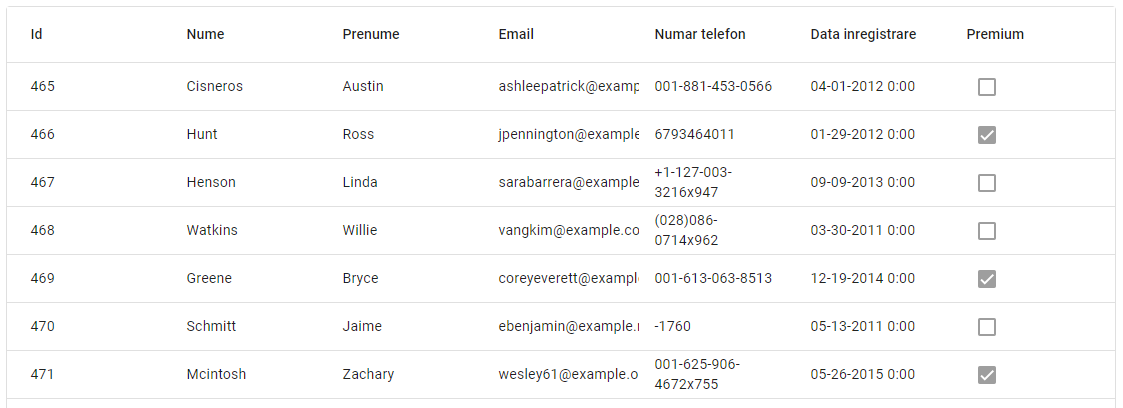
* Zbor (tabel fragmentat orizontal)



* Destinație (tabel replicat)

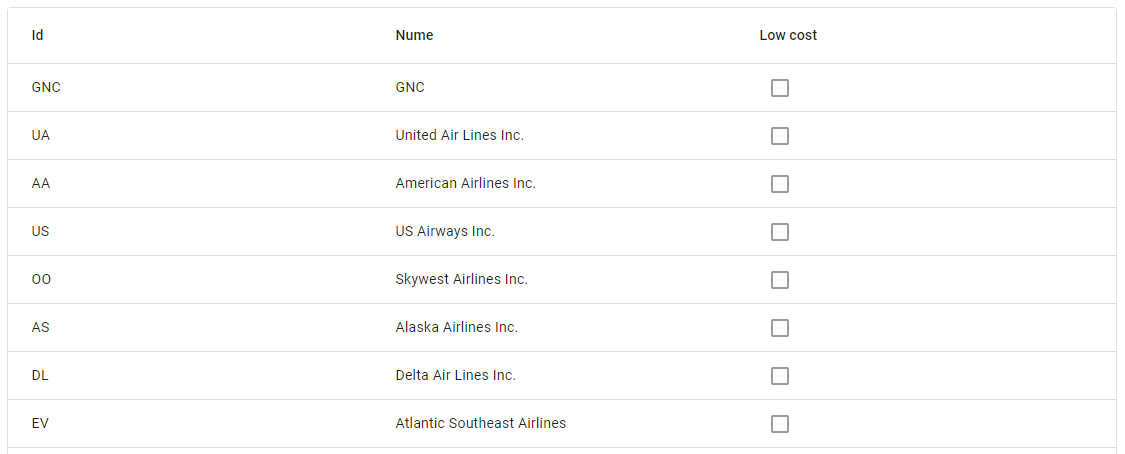


* Client (replicare și fragmentare verticală)

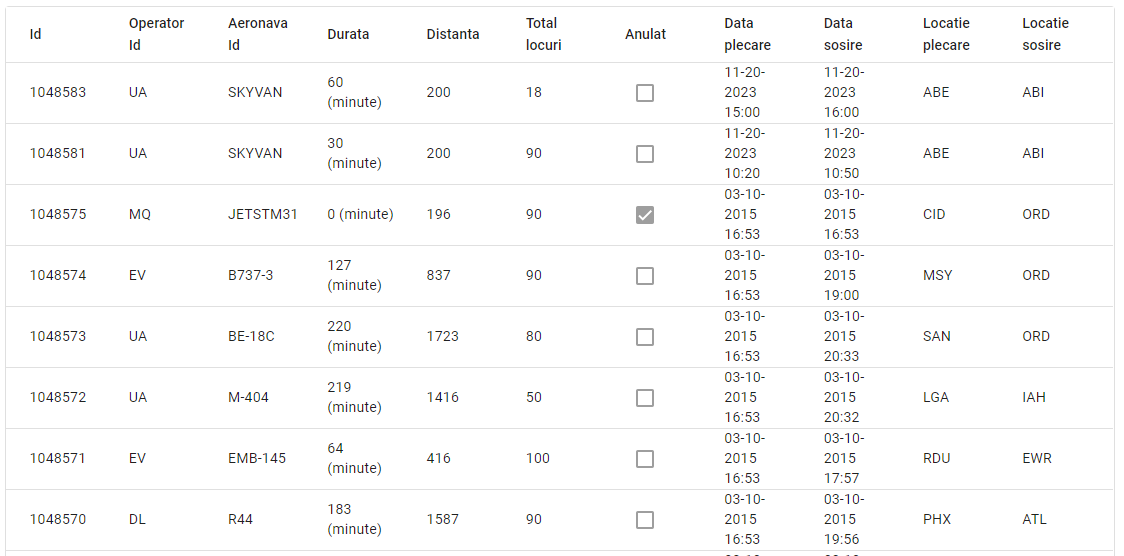


Exemplificăm vizualizarea informațiilor la nivel local pentru datele *Non low cost*:

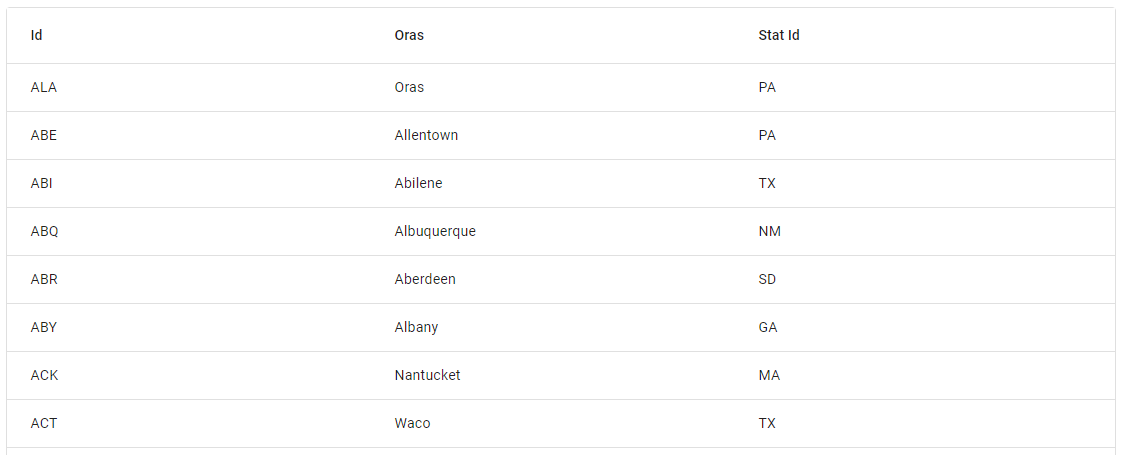
* Operator zbor (tabel fragmentat orizontal)



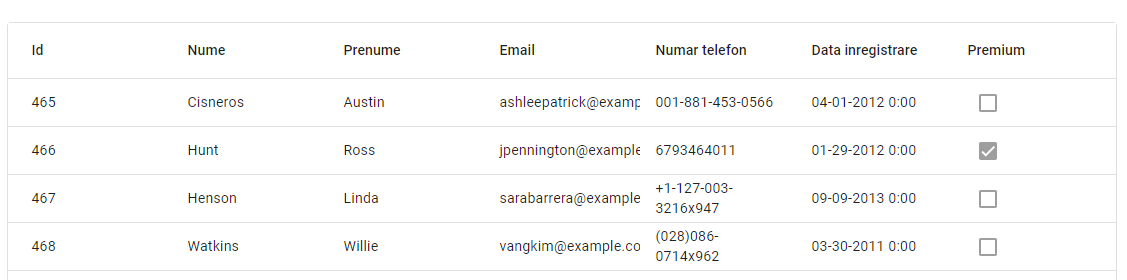
* Zbor (tabel fragmentat orizontal)



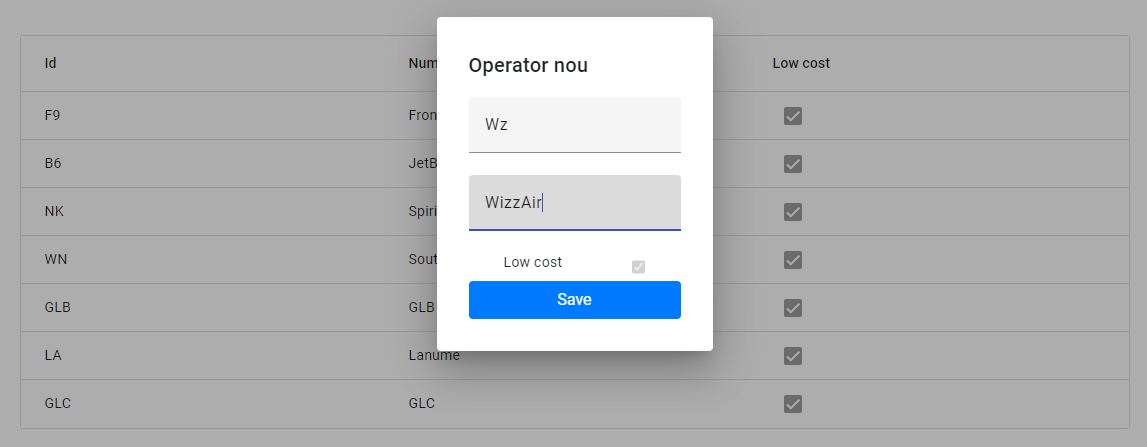
* Destinațe (tabel replicat)



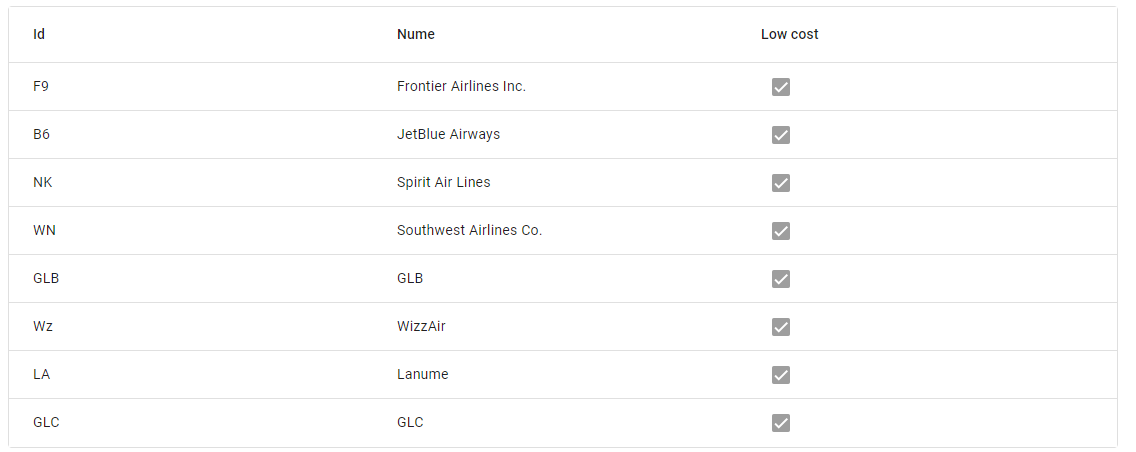
* Client (replicare și fragmentare verticală)



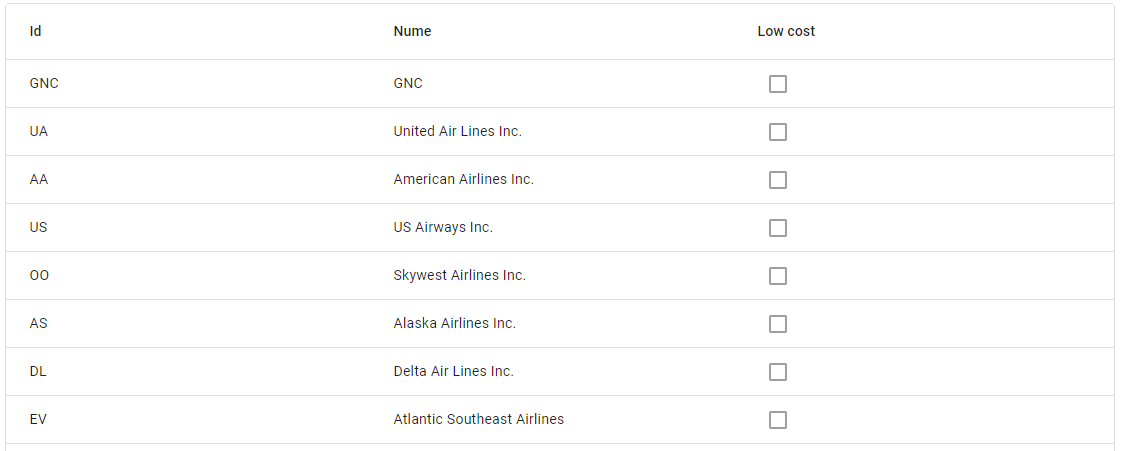
Introducerea de date la nivelul bazei de date *lowcost*:



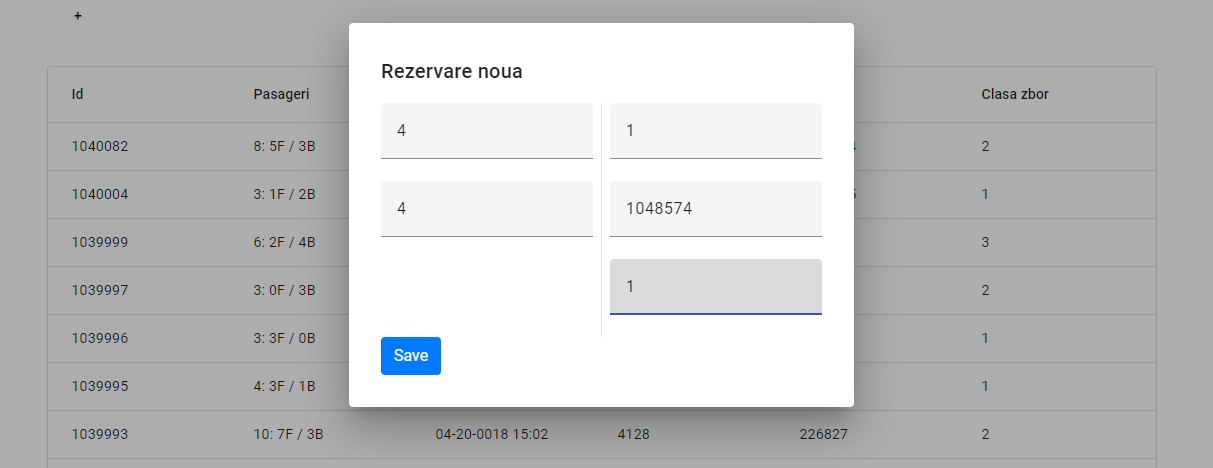
Se verifică rezultatul inserării:



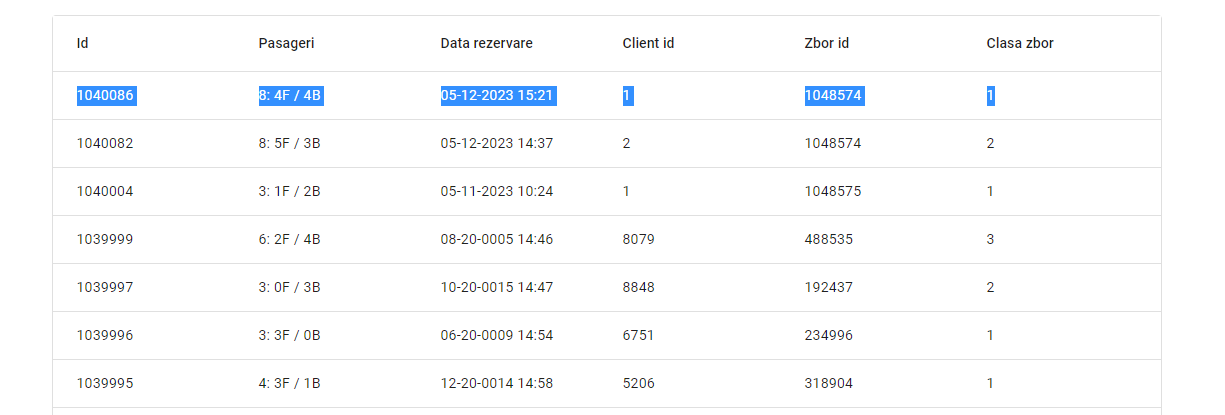
De asemenea, acesta **nu** se află inserat în baza de date corespunzătoare datelor *non-lowcost*:



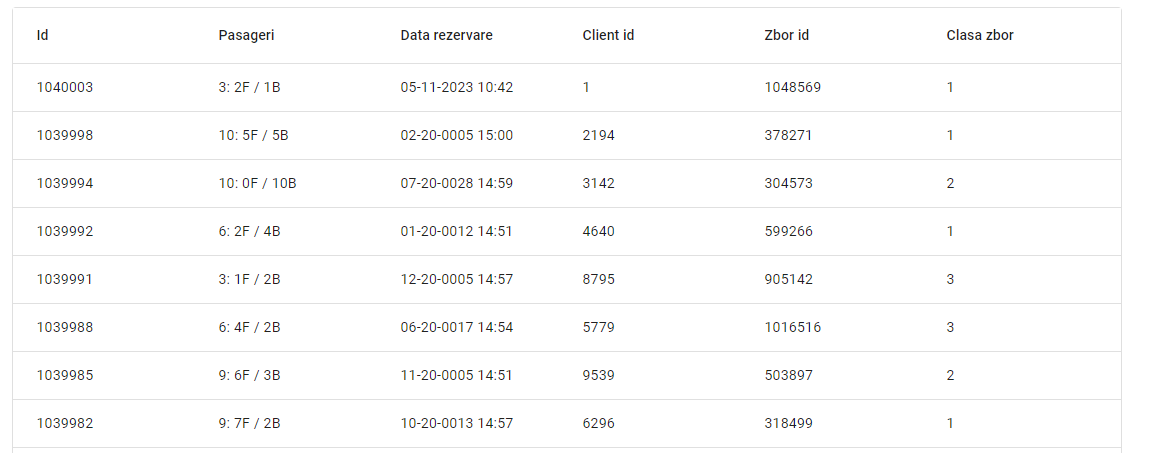
Repetăm procesul cu adăugarea unei rezervări *non-lowcost*:



Verificăm ca aceasta să fie inserată în fragmentul *non-lowcost*:

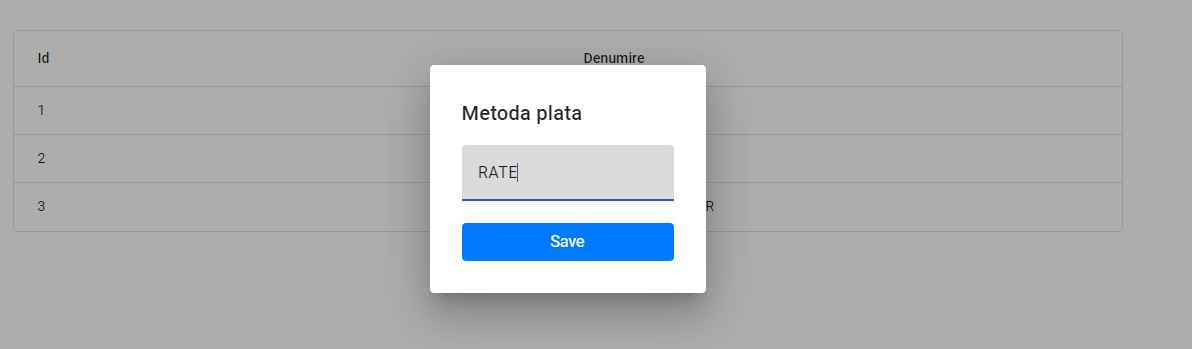
**

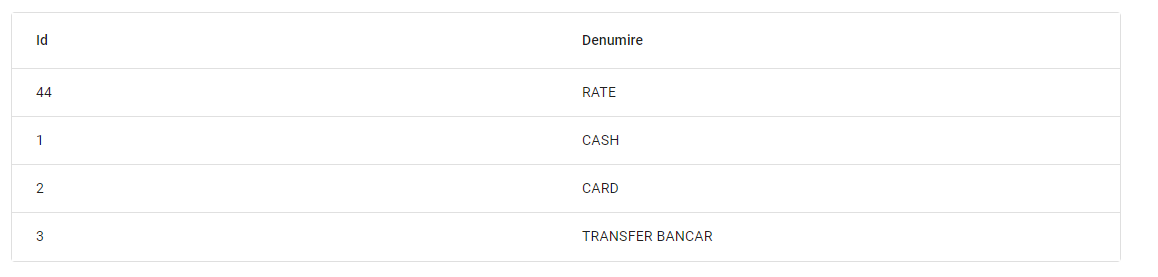
Verificăm fragmentul *lowcost*:

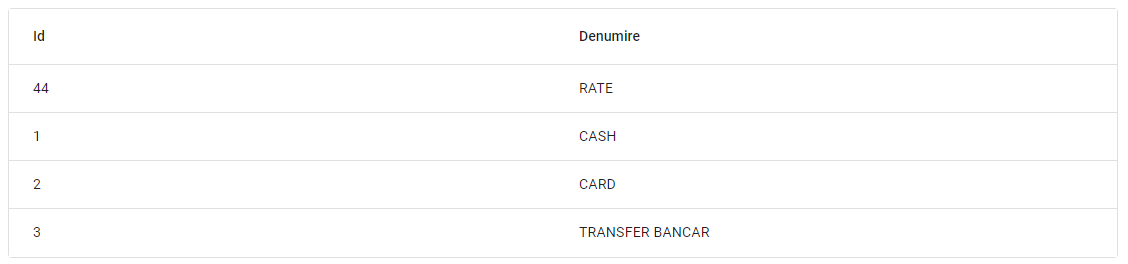


Deci, informația a fost cu succes inserată doar în fragmentul corespunzător.

De asemenea, verificăm introducerea unei noi metode de plată la nivelul bazei de date *lowcost*. Din moment ce tabela *metoda plată* este replicată, informația va fi propagată în fragmentul *non-lowcost*.



Observăm că datele au fost propagate în ambele fragmente:

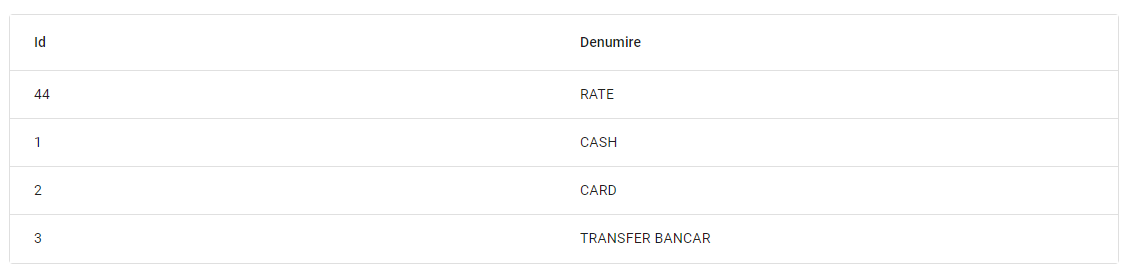


1. Vizualizarea informațiilor la nivelul bazei de date globale

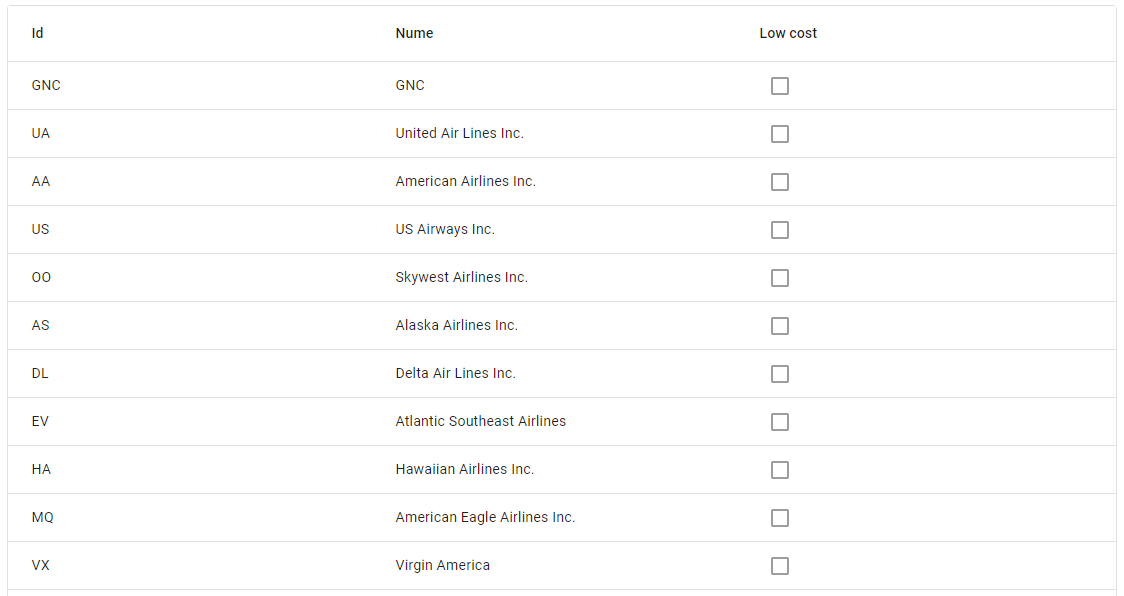
Verificăm că datele inserate pot fi vizualizate prin intermediul bazei de date globale. Accesarea ei se face prin navbar-ul marcat corespunzător:

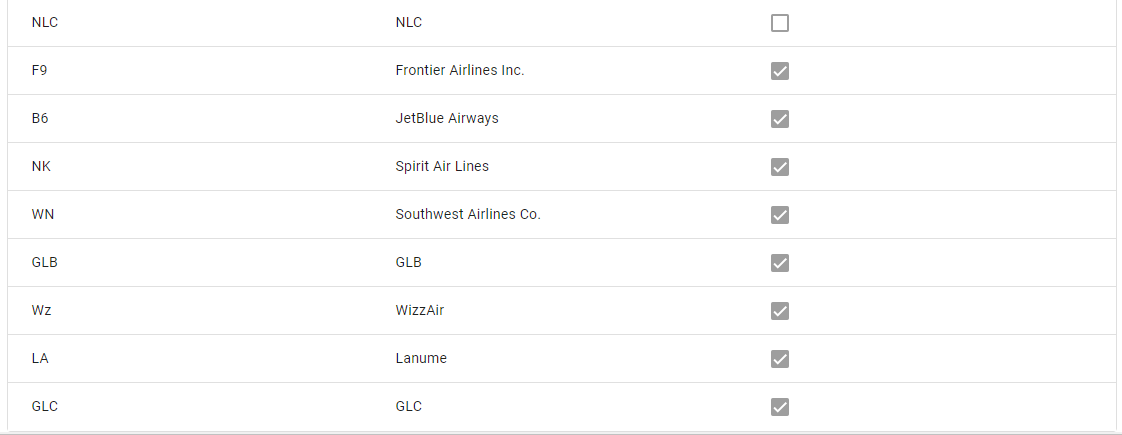


METODĂ PLATĂ (replicat):

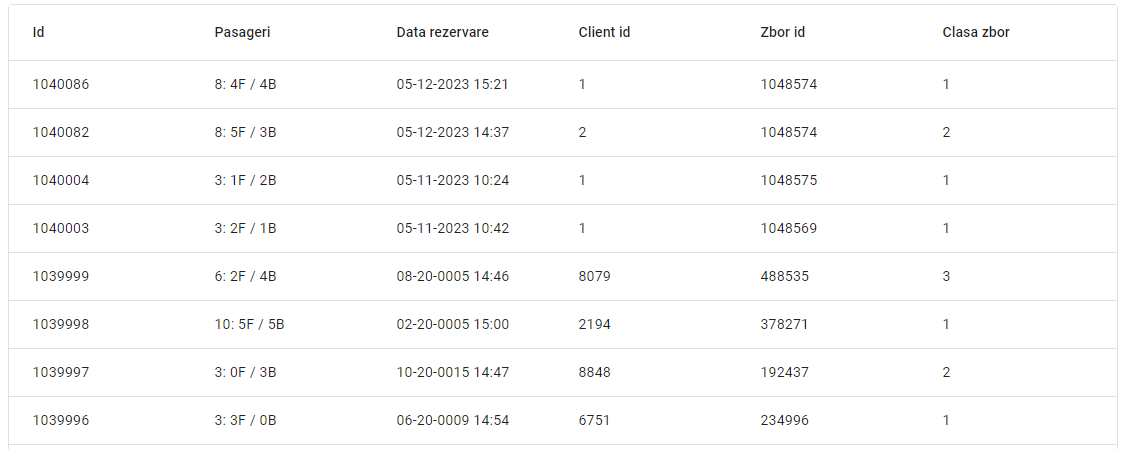


OPERATOR ZBOR (fragmentat orizontal):



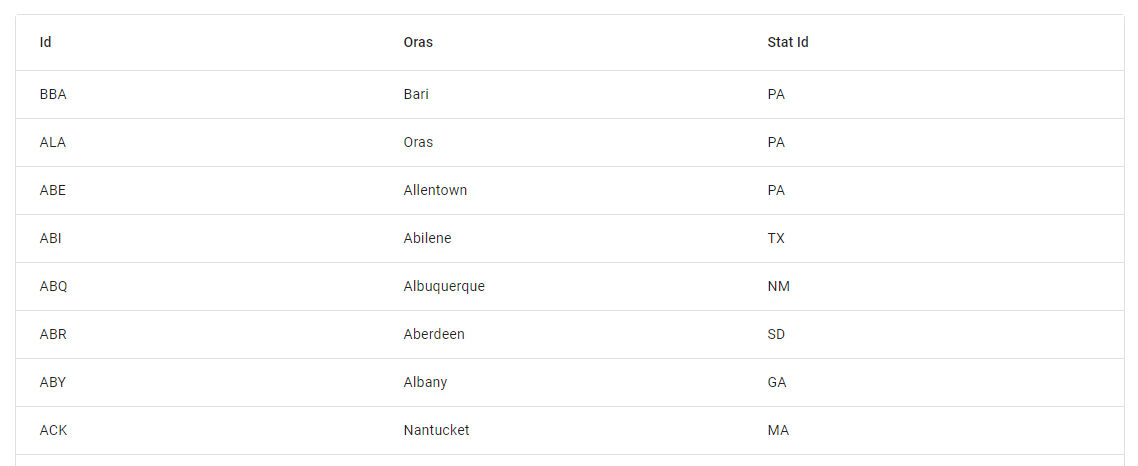


REZERVARE (fragmentat orizontal):

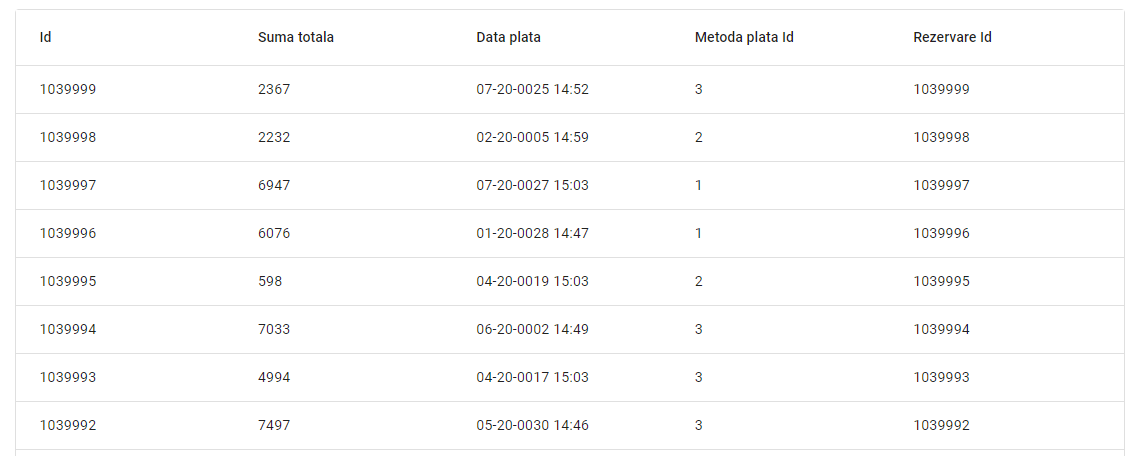


De asemenea, se poate observa că are loc agregarea datelor, în funcție de rezervările *lowcost* și *non-lowcost*.

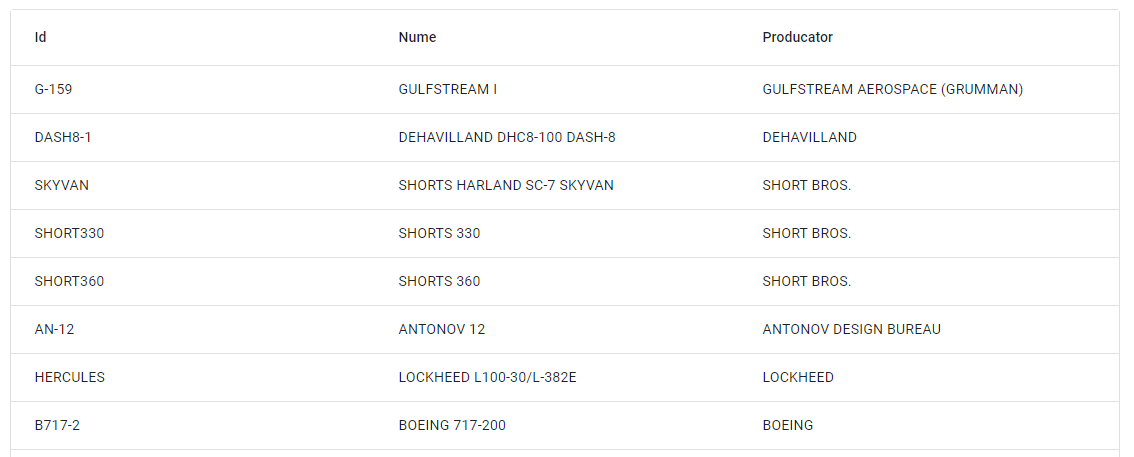
DESTINAȚIE (replicat):



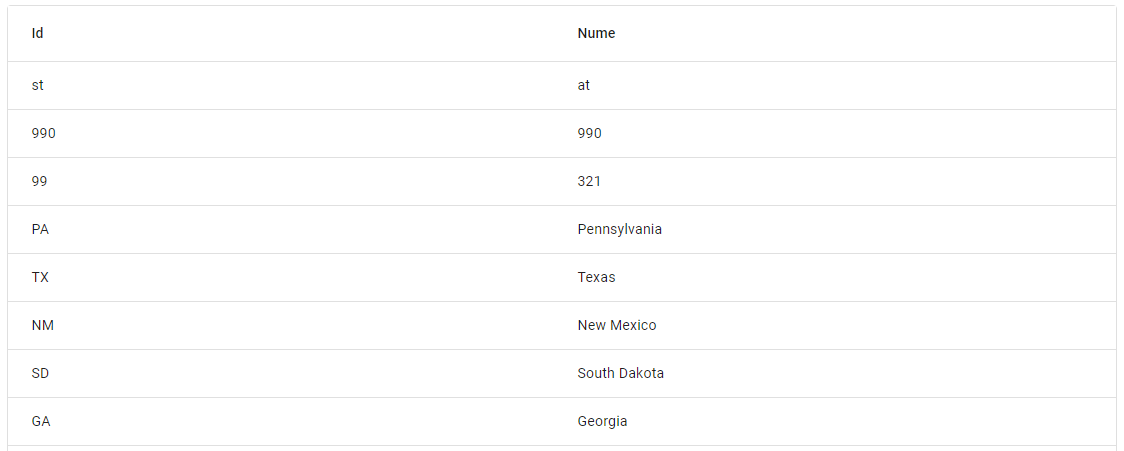
PLATĂ (fragmentat orizontal):



AERONAVE (replicat):



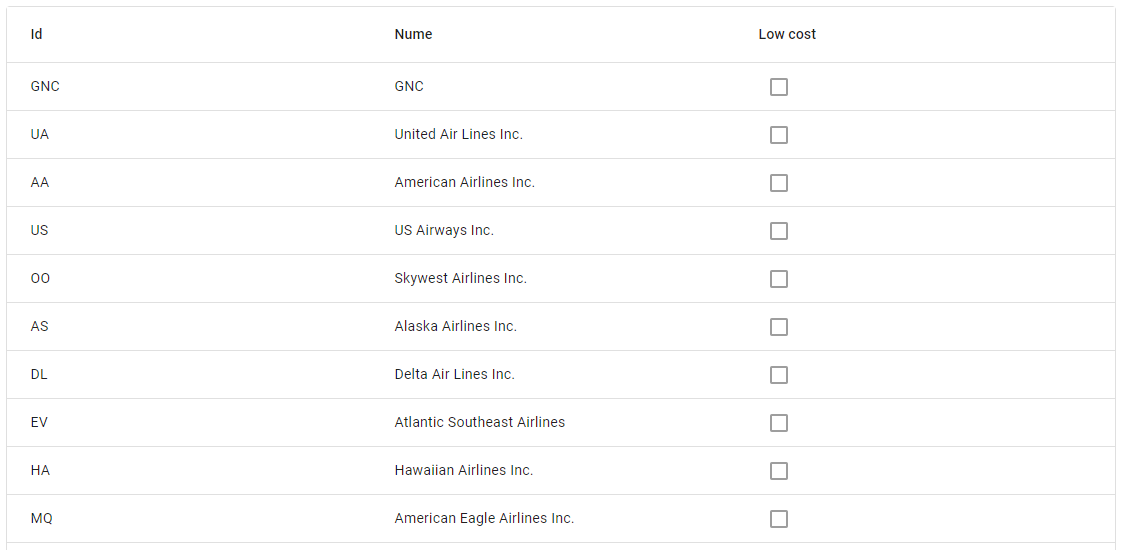
STATE (replicat):

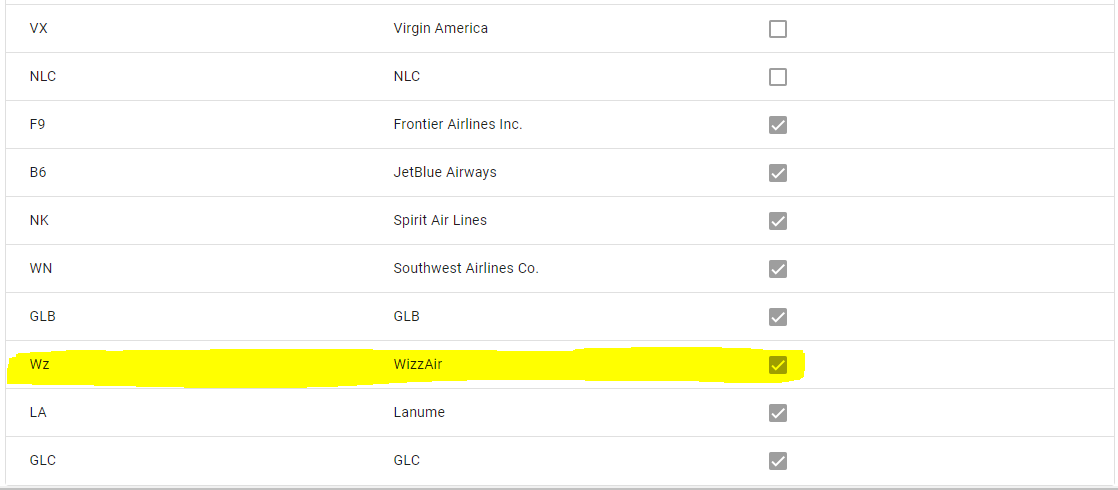


1. Vizualizarea la nivelul bazei de date globale a modificărilor aplicate asupra datelor stocate în bazele de date locale

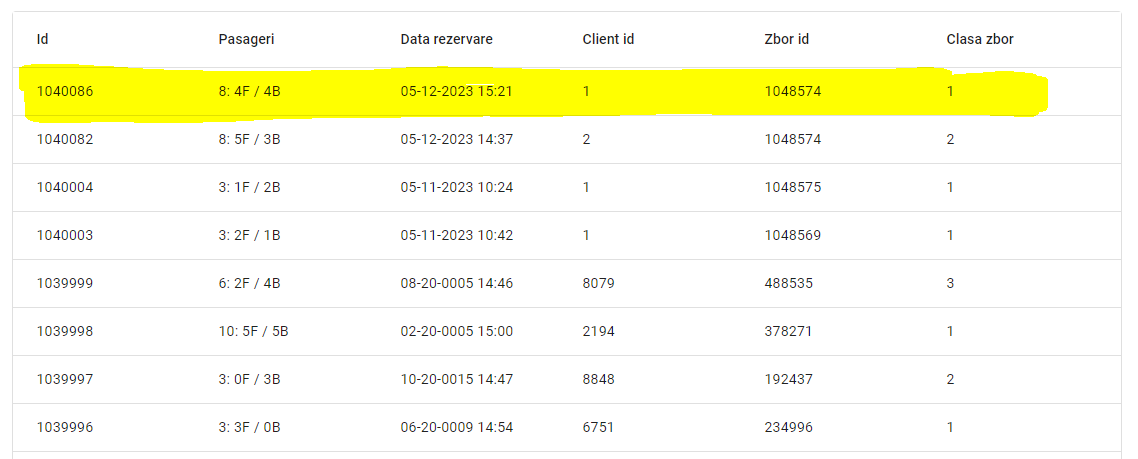
Putem verifica la nivelul bazei de date globale faptul că modificările efectuate la cerința 1 funcționează. Se verifică fiecare tabel în parte:

OPERATOR ZBOR (fragmentare orizontală):

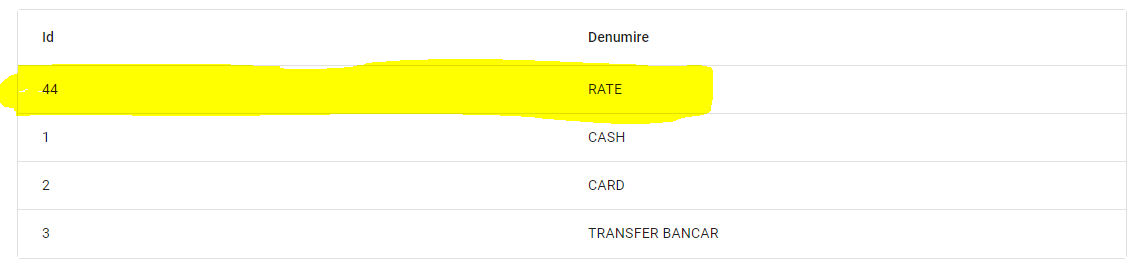




REZERVARE (fragmentare orizontală):

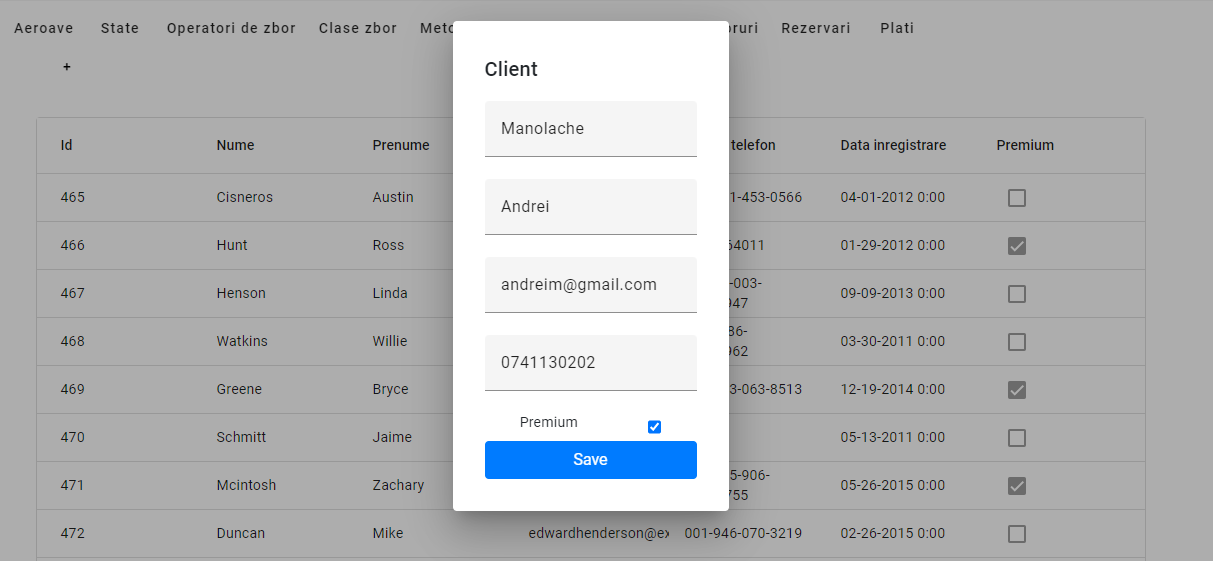


METODĂ PLATĂ (replicat):

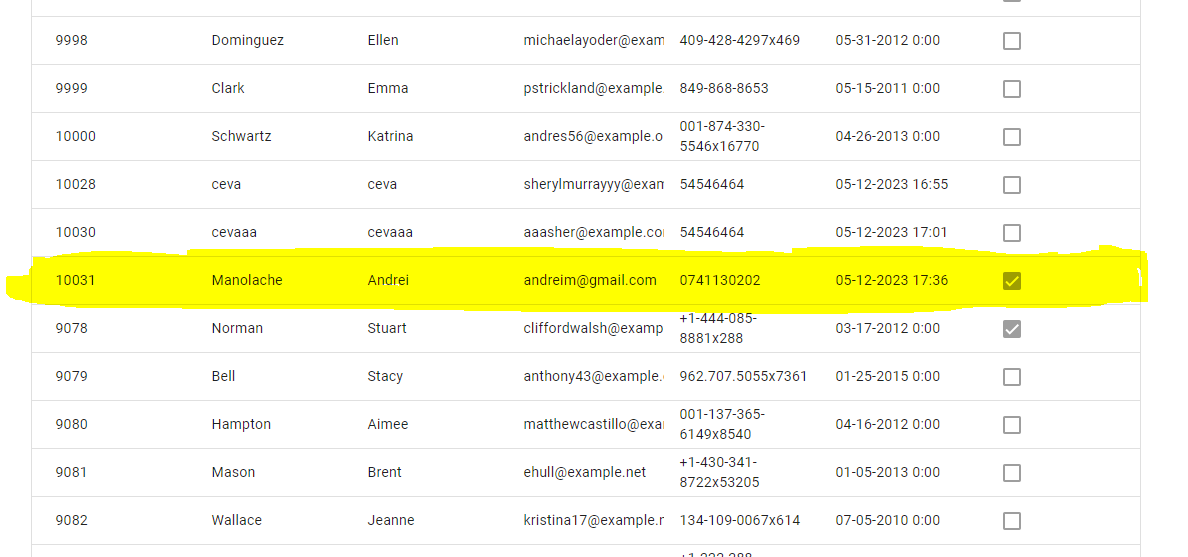


CLIENT (fragmentare verticală):

Pentru client, introducem date la nivelul bazei de date *lowcost*:

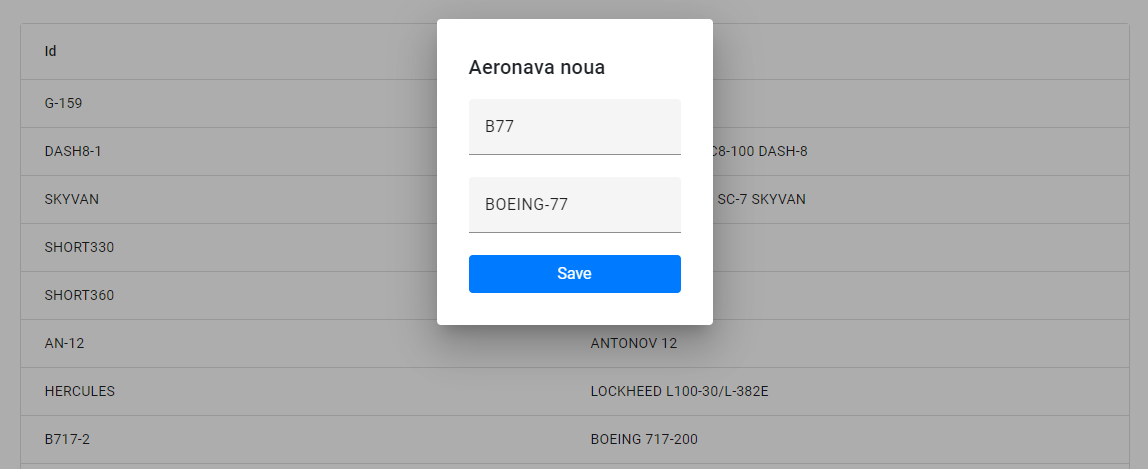


Verificăm din modulul *global* faptul că inserarea a avut loc cu succes:



1. Verificarea propagării operațiilor LMD de la nivelul bazelor de date locale în baza de date globală

Pentru aceasta, se testează prin introducerea unei noi aeronave (tabelă replicată prin intermediul vizualizării):



După ce inserarea a avut loc cu succes, se verifică cele 2 tabele replicate:

LOWCOST:



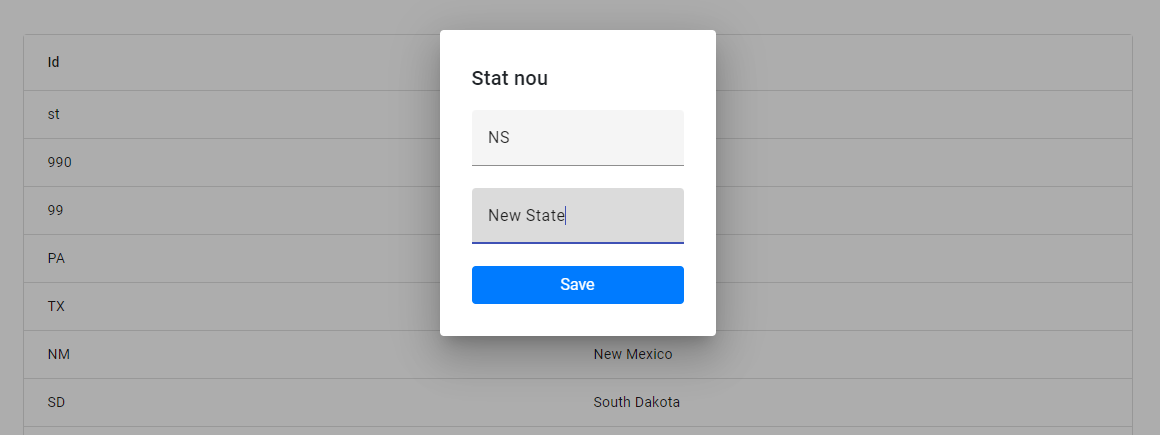
NONLOWCOST:



De asemenea, entitatea va apărea și în global:

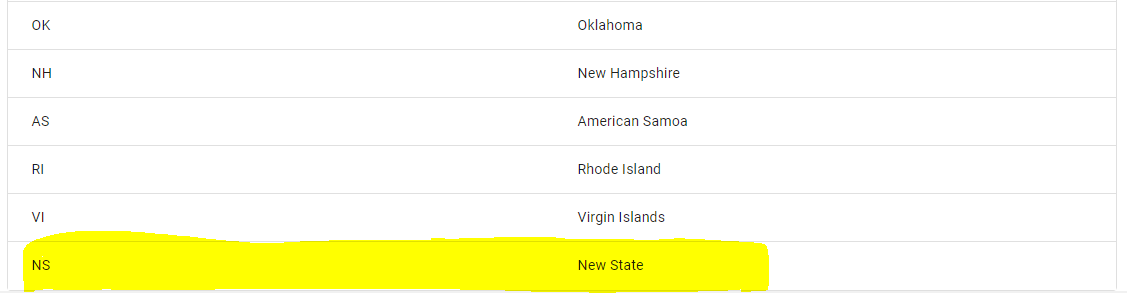


Procedăm la fel cu entitatea *Stat* (entitate replicată):

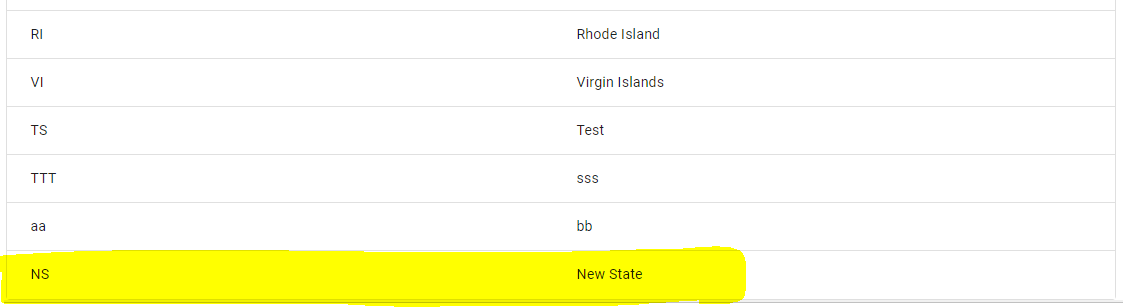


Se verifică, pe rând, faptul că a fost inserat în cele 2 baze de date diferite:

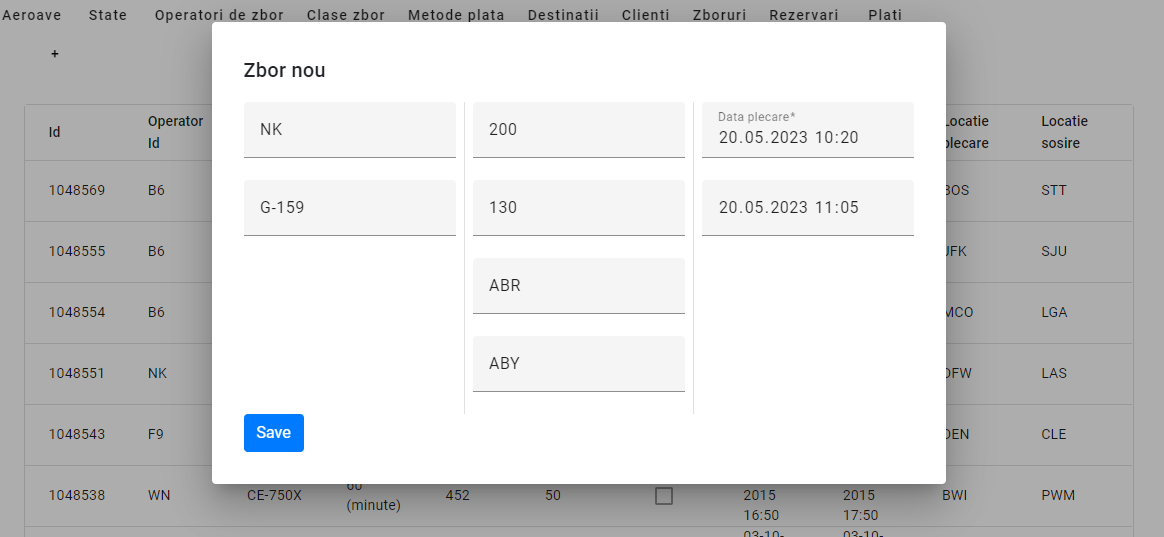
LOWCOST:



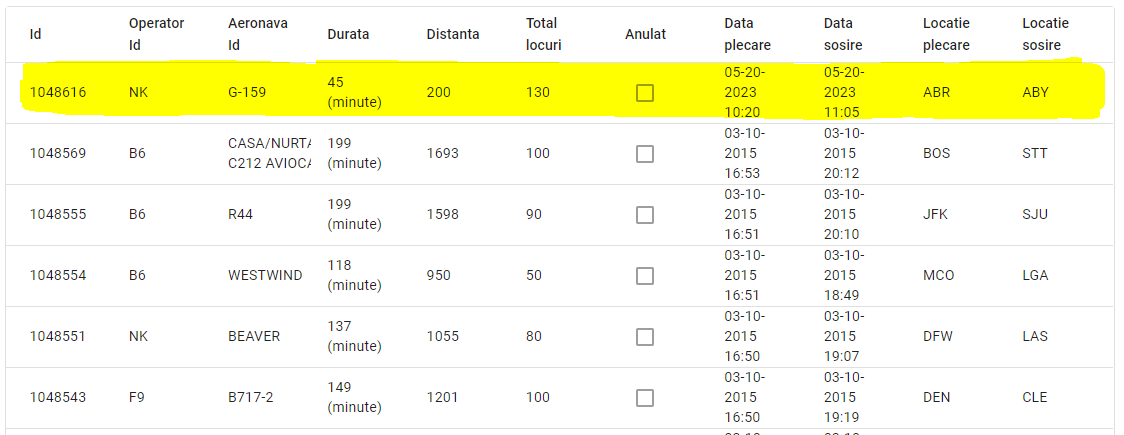
NON-LOWCOST:



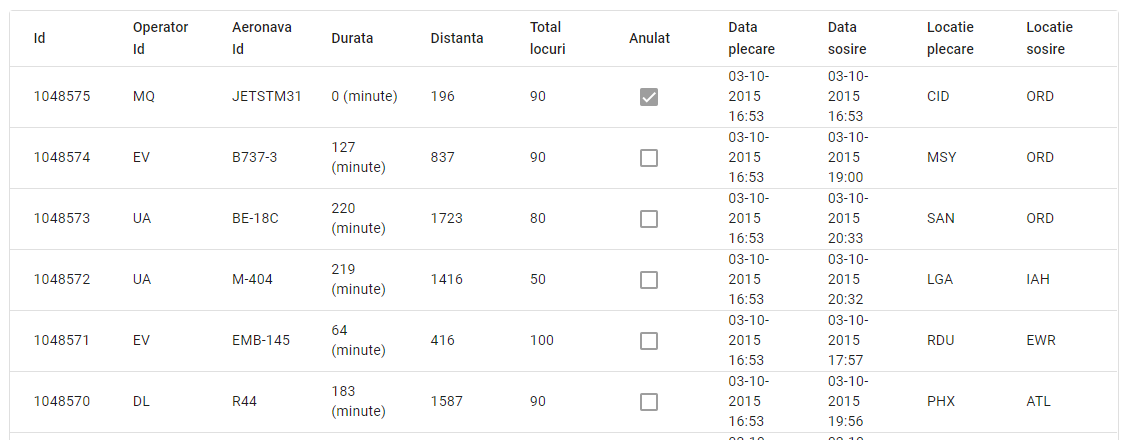
Pentru entitatea *Zbor* (enitate fragmentată orizontal), se creează un zbor *lowcost*:



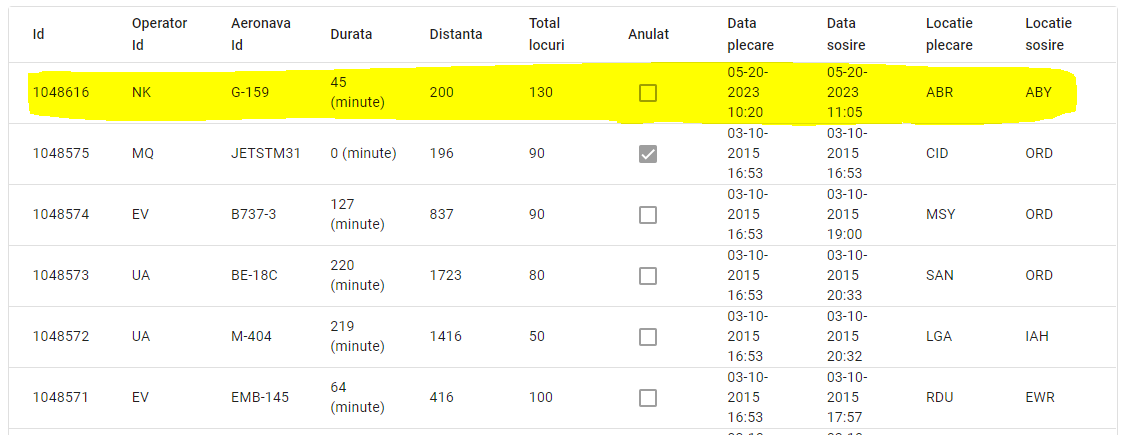
Se verifică existența lui în baza de date corespunzătoare fragmentării *lowcost*:



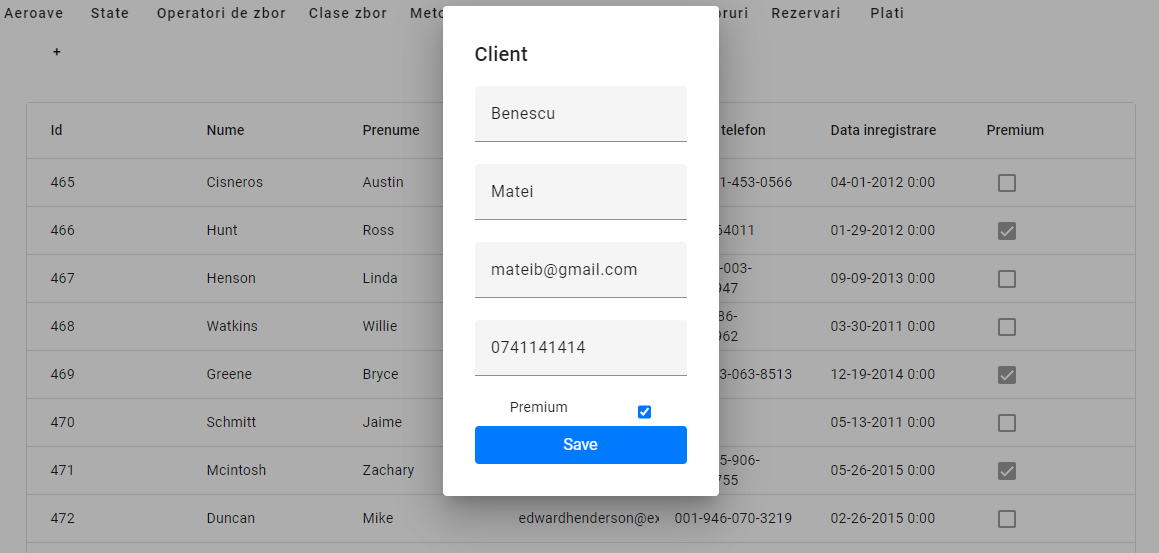
Se observă această nouă înregistrare nu se află în baza de date corespunzătoare datelor *non-lowcost*:



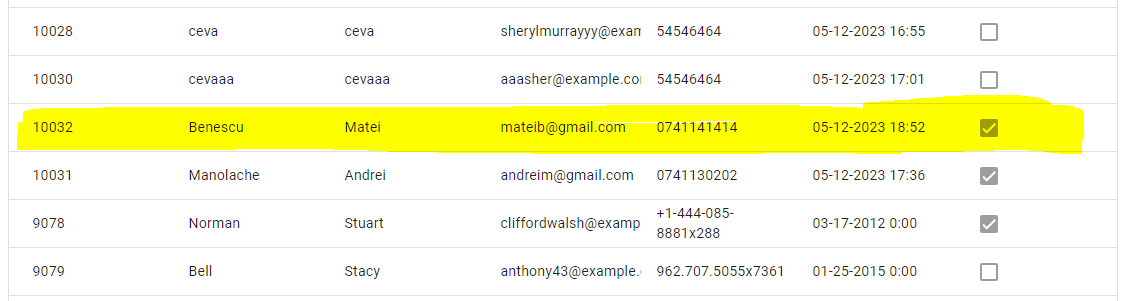
În baza de date globală se pot vedea toate înregistrările, inclusiv aceasta:



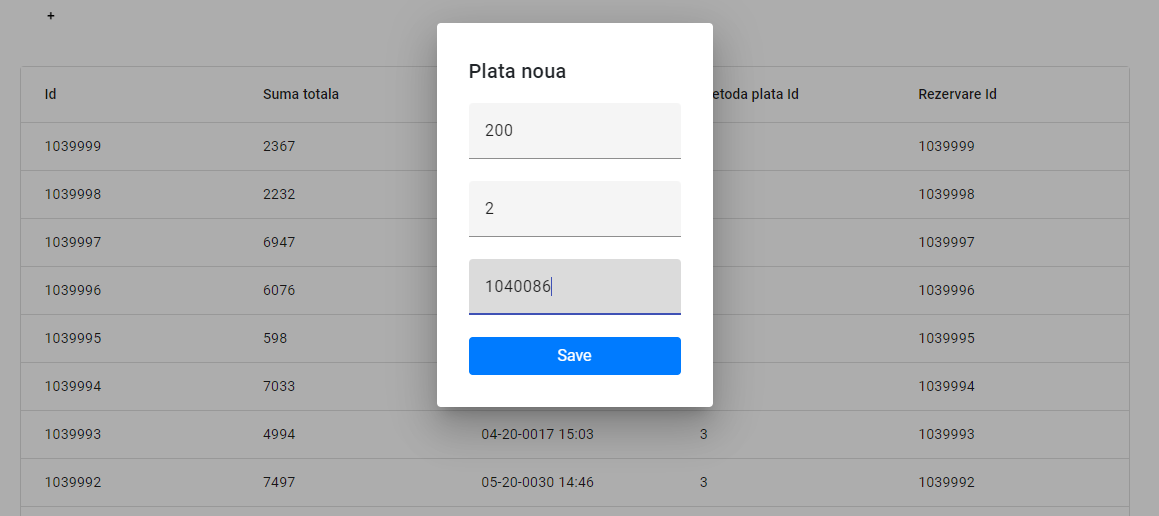
Repetăm scenariul cu entitatea *Client* pentru care există replicare și fragmentare verticală:



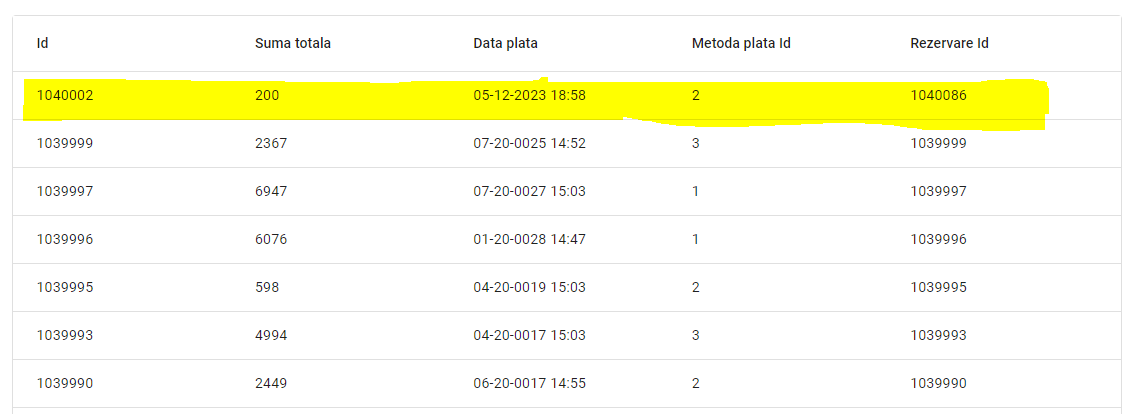
Se verifică existența noii înregistrări:



Pentru entitatea *Plata* (fragmentare orizontală) se introduce o nouă înregistrare:



Aceasta este adăugată în baza de date corespunzătoare în funcție de rezervarea pentru care se face plata. Dacă rezervarea este pentru un zbor *lowcost*, atunci va fi inserată doar în baza de date *lowcost*, altfel va fi adăugată în *non-lowcost*. În cazul de față, rezervarea cu id-ul *1040086* este una de tip *non-lowcost*.



Se verifică că aceasta nu este inserată în baza de date destinată plăților *lowcost*:

