

**UNIVERSITATEA DIN BUCUREȘTI**

FACULTATEA DE

MATEMATICĂ ȘI INFORMATICĂ

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

*Modulul analiză*

**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:

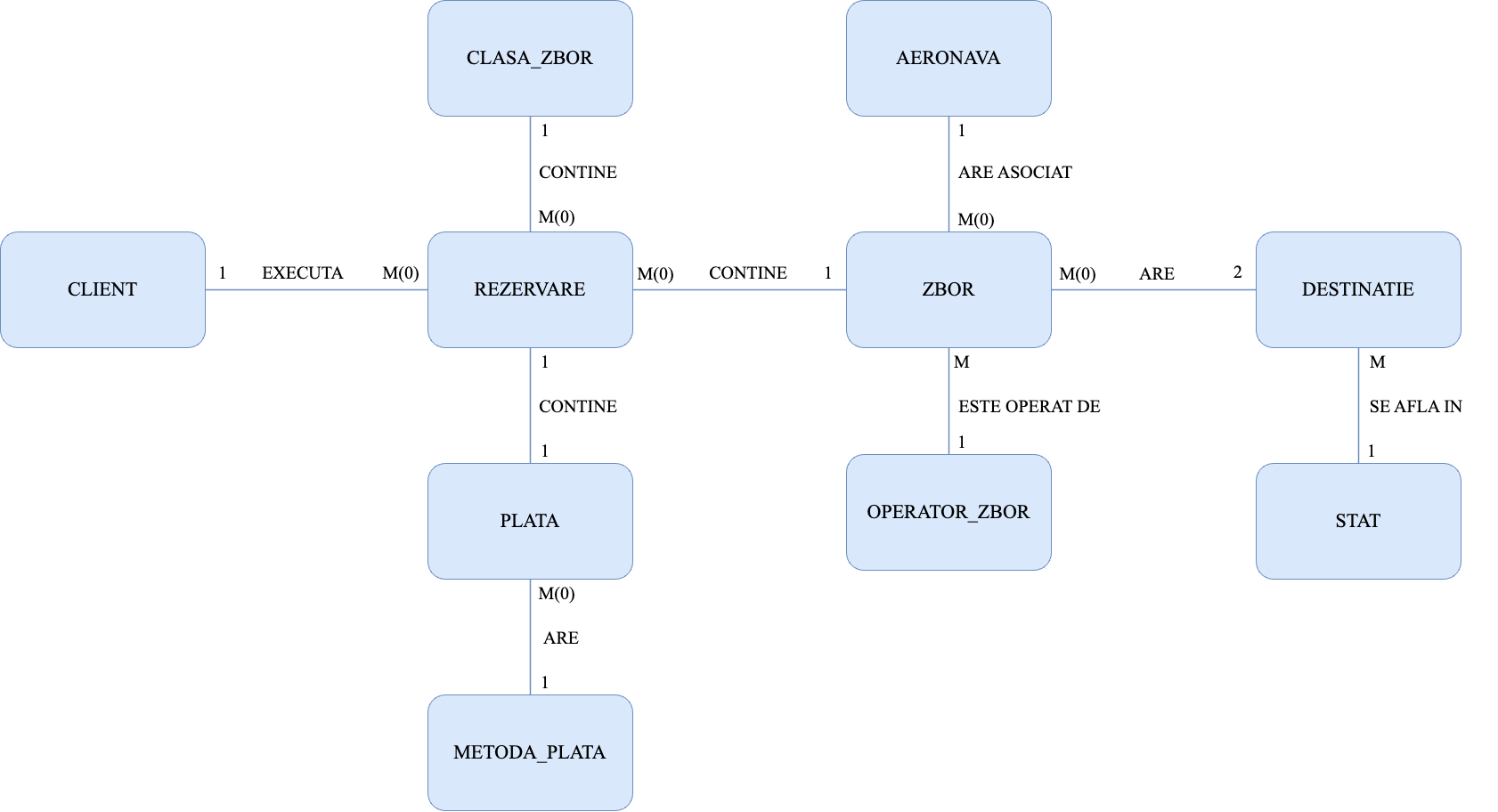
* Adaugă coloana zbor\_id, creată incremental, pentru a o folosi în continuare cu ușurintă
* Generează coloana total\_locuri, cu valori între 50 și 100
* Modifică coloanele data\_plecare și data\_sosire de tip timestamp, pentru a conține și ora
* Adaugă o cheie externă spre aeronave

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.

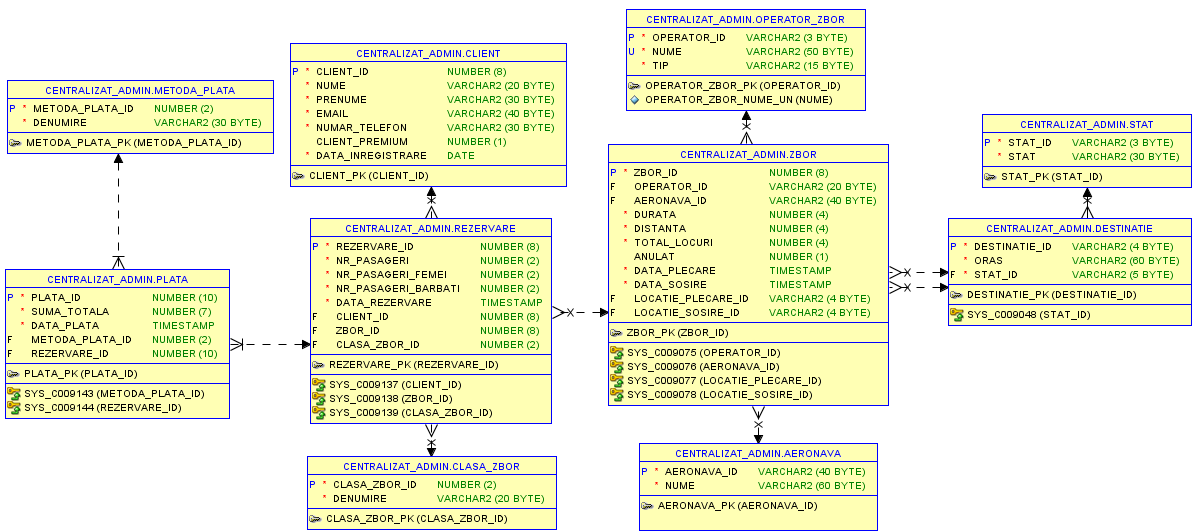
Rezervările clienților au fost generate în totalitate folosind un script de python și conțin: id-ul, numărul total de pasageri conținut de rezervare, numărul de pasageri care sunt femei, respectiv bărbați, data rezervării, id-ul clientului, id-ul zborului, clasa la care au fost rezervate locurile și 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.

2. Diagramele bazei de date OLTP inițiale

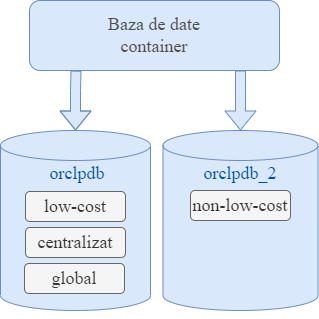
* Diagrama entitate - relație a bazei de date OLTP



* Diagrama conceptuală a bazei de date OLTP



3. 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.

4. 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

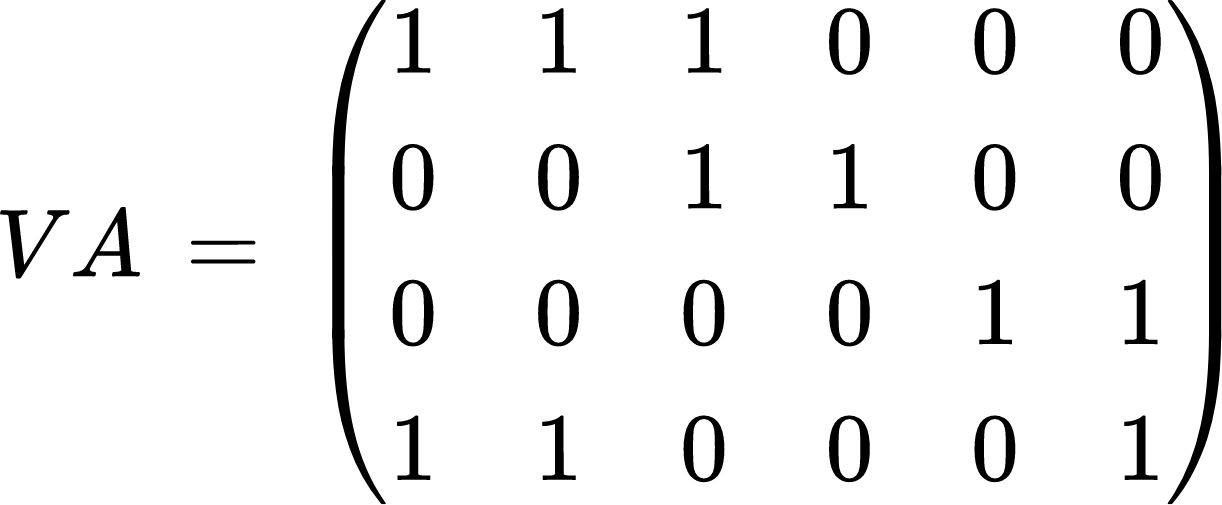
A3 = email

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

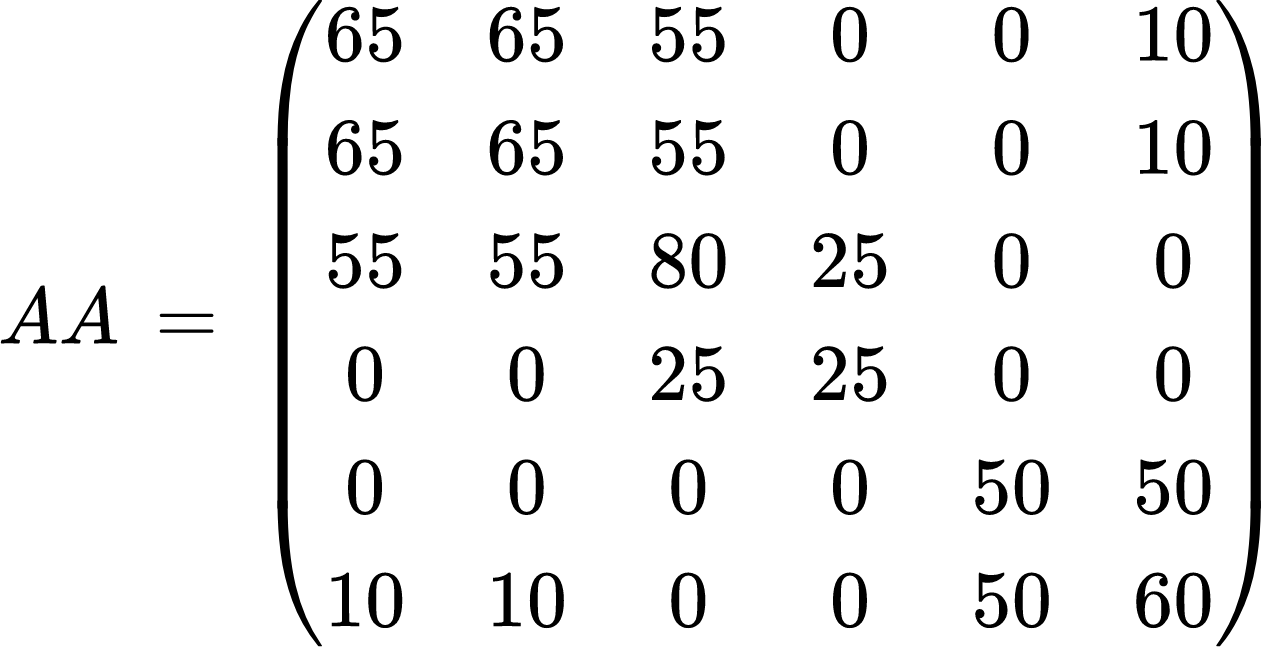
af(A4, A6) = 0

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

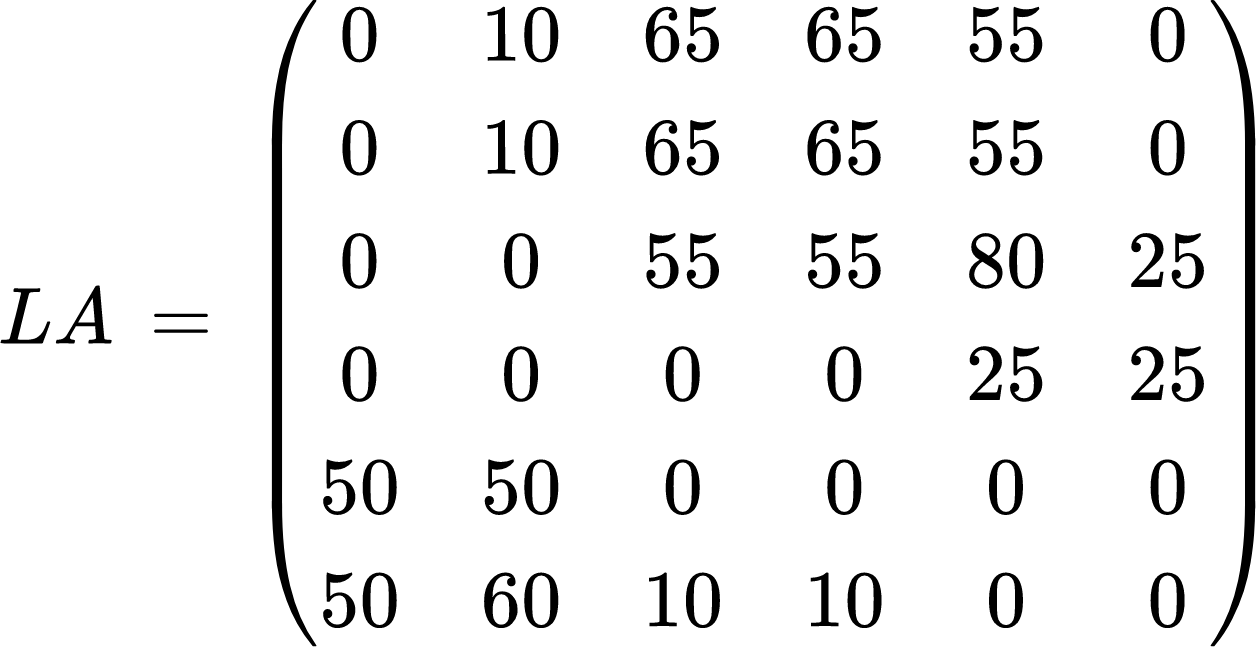
bond(A4, A6) = 0

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)

5. 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; |



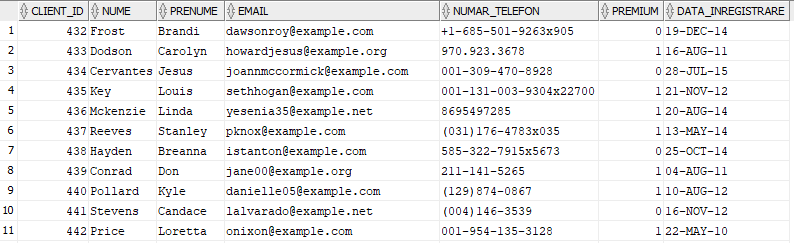
Toate tabelele rezultate sunt goale, ceea ce indică corectitudinea fragmentării.

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.

6. Replicarea

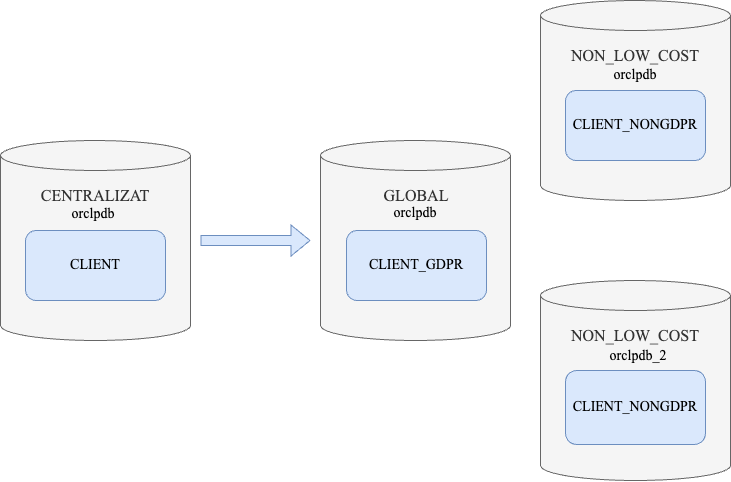
Tabelele care trecut prin procesul de replicare sunt METODA\_PLATA, CLASA\_ZBOR, DESTINATIE, STAT, AERONAVA și CLIENT\_NONGDPR. Acestea sunt tabelele care nu au fost fragmentate orizontal și trebuie să existe câte o copie pe ambele stații de lucru, cea corespunzătoare companiilor aeriene *low-cost,* 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.*

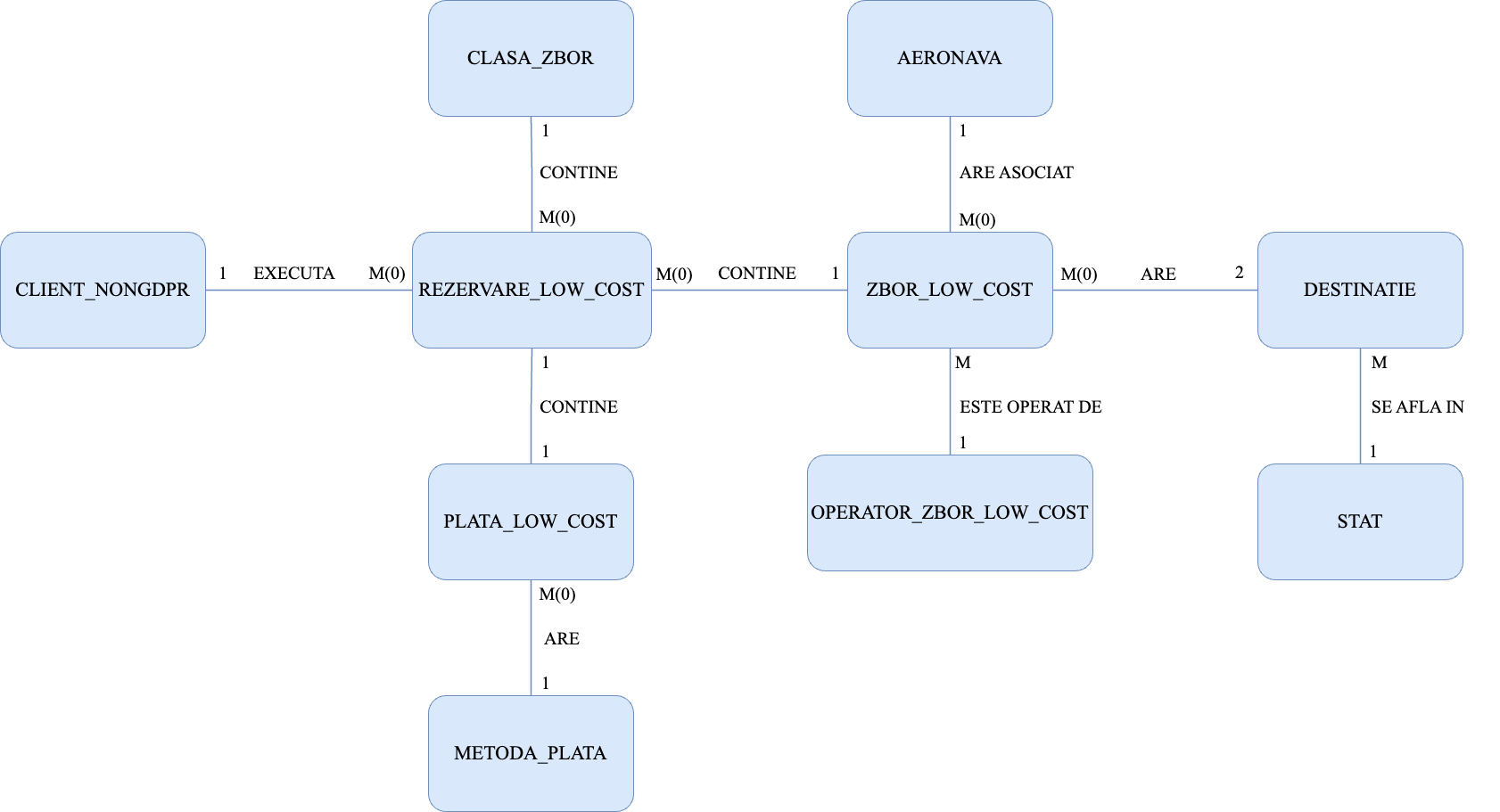
Pentru replicarea tabelei AERONAVA a fost construită o vizualizare materializată în baza de date *non-low-cost*, împreună cu tabela *log* asociată. Pentru optimizare, fost aleasă o actualizare de tip *refresh fast,* pentru a realiza actualizări mai rapide, tabelul *log* înregistrând ultimele schimbări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

7. Rezultatele fragmentării și replicării

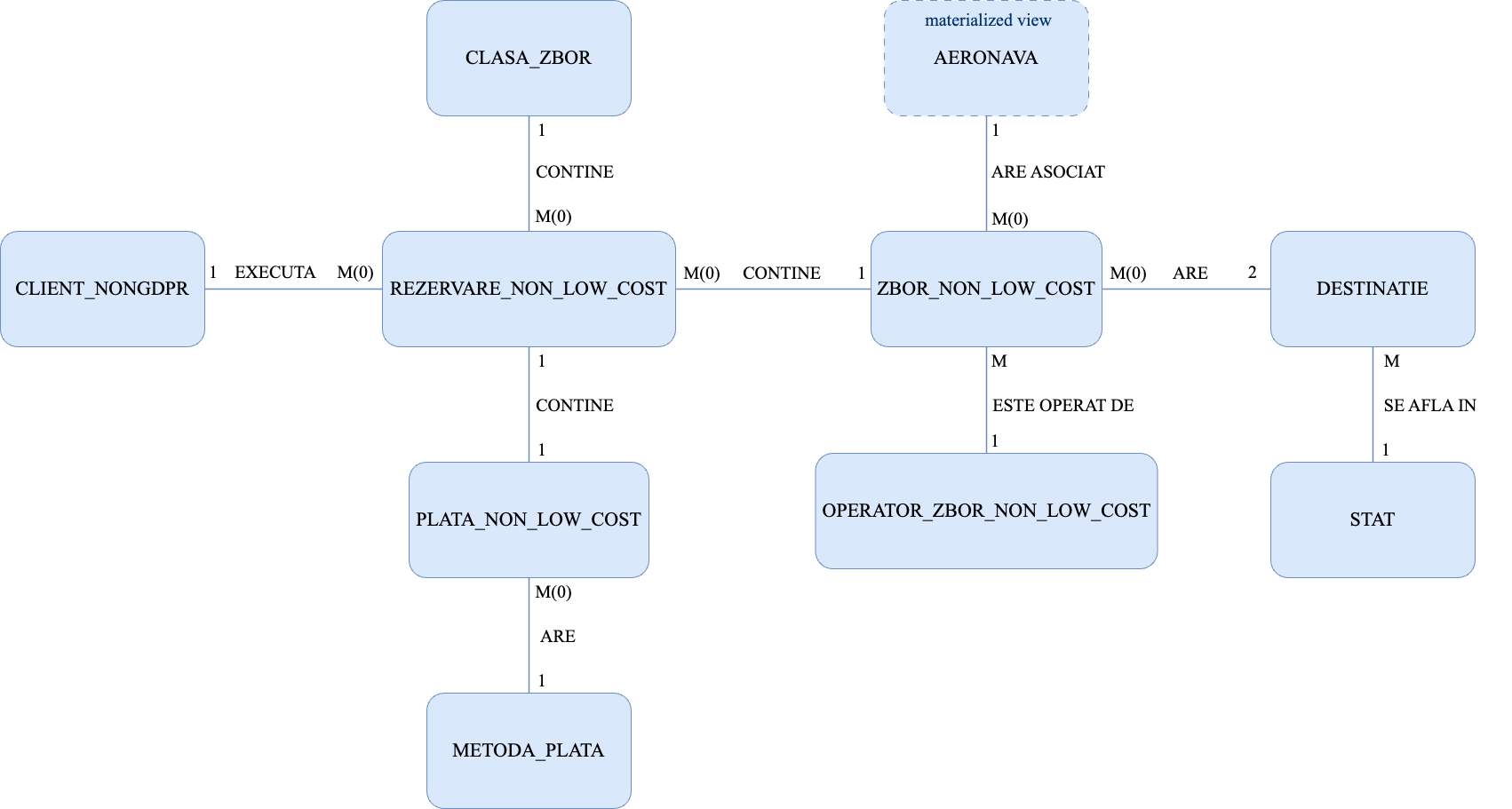
În urma fragmentării verticale, tabela CLIENT va avea următoarea structură:



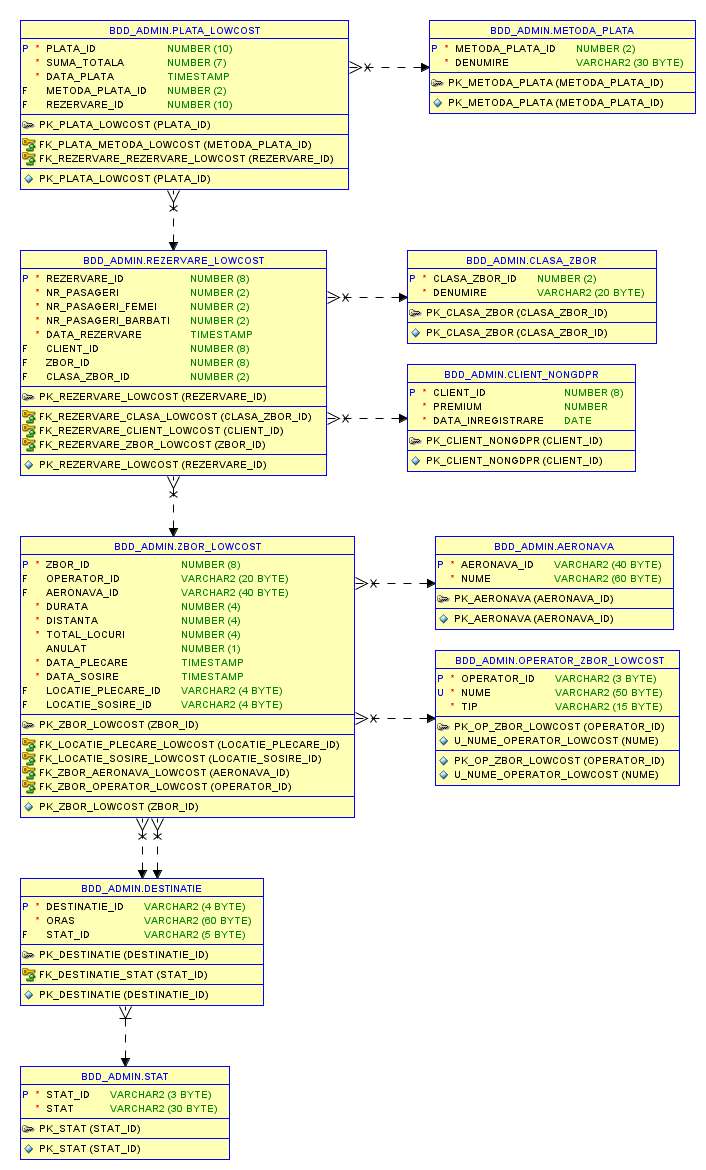
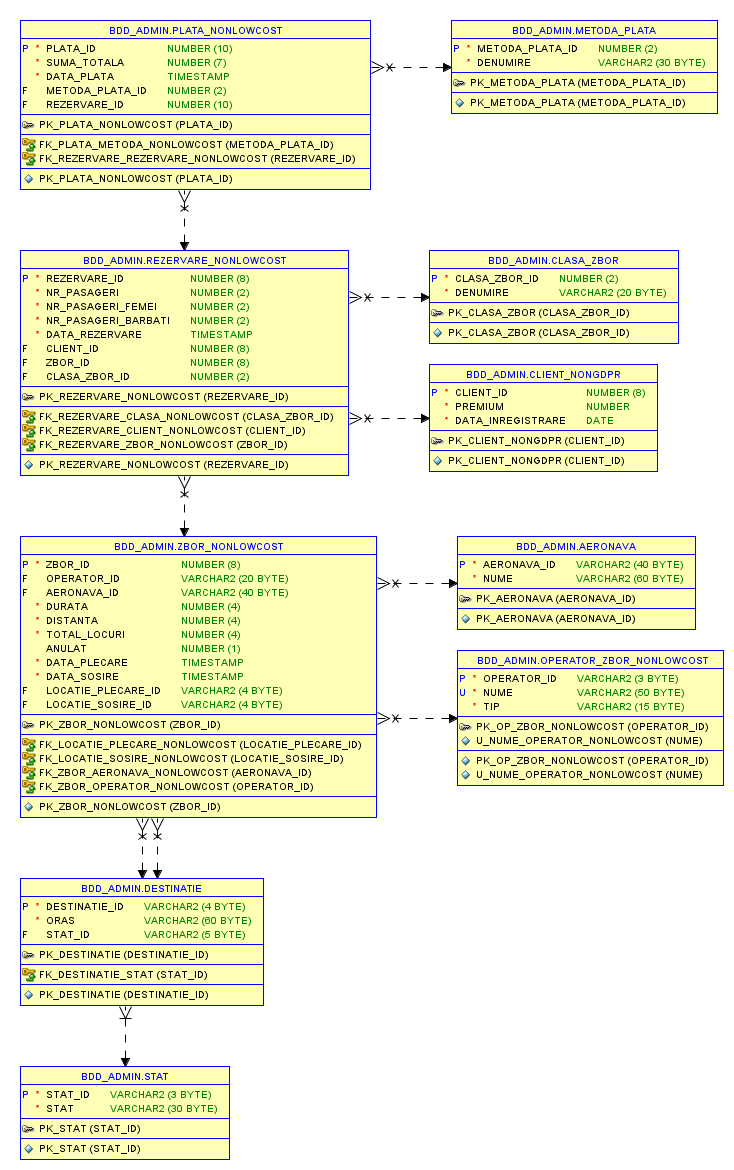
Bazele de date corespunzătoare celor două stații de lucru:

* orclpdb *(lowcost)*

* orclpdb\_2 *(lowcost)*



8. Schemele conceptuale locale corespunzătoare bazelor de date locale

* *Low cost*
* *Non low cost*

9. Constrângerile ce trebuie îndeplinite de model

* **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.

1. **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.

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.

* **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.

* **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.

* **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.

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.

* **NOT NULL**

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

10. Formularea unei cereri SQL complexe

*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.