PROIECT BAZE DE DATE

TEMA: GESTIUNEA UNUI LANȚ DE RESTAURANTE

*Lect. Dr. Marin Letiția Ana*

*Student: Sora Andreea-Ioana*

*Seria 13, Grupa 134*

*Anul I, Semestrul II*

*CUPRINS*

[Cerința 1 – Modelul real, reguli de funcționare](#model).............................................3

[Cerința 2 – Prezentarea constrângerilor](#constrangeri).......................................................4

[Cerința 3 – Descrierea entităților](#entitati).................................................................4

[Cerința 4 – Descrierea relațiilor](#relatii)...................................................................5

[Cerința 5 – Descrierea atributelor](#atribute)..............................................................10

[Cerința 6 – Diagrama entitate-relație](#er).........................................................14

[Cerința 7 – Diagrama conceptuală](#diagconceptuala).............................................................15

[Cerința 8 – Schemele relaționale](#scheme)................................................................15

[Cerința 9 – Normalizarea până la forma normală 3](#normalizare)....................................16

[Cerința 10 – Crearea tabelelor în SQL și inserarea datelor](#creare_tabele).........................18

[Cerința 11 – Cereri SQL](#cereriSQL)...........................................................................24

[Cerința 12 – Actualizare sau suprimare a datelor folosind subcereri](#actualizare)...........27

[Cerința 13 – Secvență utilizată în inserarea datelor](#secvente)....................................28

[Cerința 14 – Outer-join/Division](#division)...............................................................29

[Cerința 15 – Optimizarea unei cereri](#optimizare)........................................................30

[Cerința 16.1 – Normalizarea BCNF, FN4, FN5](#fn45bc)..........................................33

[Cerința 16.2 – Denormalizare](#denormalizare)...................................................................36

[Bibliografie](#bibliografie)..............................................................................................37

1. Descrierea modelului real, a utilității acestuia și a regulilor de funcționare.

În acest proiect vom analiza gestiunea unui lanț de restaurante. Aceasta va ajuta la o mai bună funcționare, la posibilitatea de a ține evidența comenzilor plasate de către fiecare client către fiecare restaurant și la gestionarea evenimentelor ce pot avea loc în fiecare dintre locațiile noastre. Regulile de funcționare pentru acest model sunt următoarele:

* În restaurante se pot organiza mai multe evenimente cărora le corespunde un tip de eveniment și, în funcție de tip, anumite facilități;
* Pentru evenimentele care pot avea loc (nuntă, botez, majorat, petreceri de weekend, petreceri de 8 martie etc) se pot face mai multe rezervări. Vom reține inclusiv evenimentele pentru care nu s-a făcut nicio rezervare până la momentul actual;
* Un eveniment are o dată de programare, dar poate avea loc simultan în mai multe din locațiile noastre în cazul în care acest lucru este posibil (spre exemplu, evenimente de 8 martie);
* Angajații au un anumit job și un istoric unde vom ține evidența posturilor pe care aceștia le-au avut în restaurant. Data de angajare a fiecărui salariat va fi prima data când acesta s-a angajat. În cazul în care cineva își schimbă job-ul, acest lucru se va consemna în istoric. Vom presupune că un salariat, o dată angajat într-unul din restaurante, va putea schimba job-ul doar în cadrul aceluiași restaurant;
* Restaurantele pot avea o adresă și un program de funcționare;
* Fiecare restaurant din lanțul nostru dispune de același meniu;
* Clienții pot face rezervări pentru evenimente și pot plasa comenzi, având de asemenea un istoric de comenzi;
* La fiecare rezervare se va reține numărul de locuri pentru care clientul solicită acest lucru. Acesta poate face rezervare pentru nuntă/botez specificând numărul de invitați sau pentru evenimentele de alt tip/comune ce permit rezervări pentru un număr limitat de persoane, ținând cont de numărul maxim de persoane stabilit de fiecare restaurant;
* Comanda cuprinde diverse preparate, într-o anumită cantitate;
* Se vor reține o singură dată informațiile despre clienți alături de prima comanda (dacă acesta solicită o comanda înainte de a face o rezervare, altfel se vor reține doar datele de contact), urmând ca pe viitor, celelalte comenzi să se consemneze în istoricul comenzilor. Pentru fiecare comanda, clienții oferă o nota/recenzie.

1. Prezentarea constrângerilor (restricții, reguli) impuse asupra modelului.

Constrângerile impuse asupra modelului sunt:

* Restaurantele pot organiza mai multe evenimente, însă nu este obligatoriu acest lucru;
* Un angajat poate lucra într-un singur restaurant; pot exista restaurante fără angajați. De asemenea, pot exista angajați care nu au restaurantul setat;
* Un angajat nu poate să primească două job-uri diferite în aceeși zi;
* Salariul trebuie să fie cuprins între limitele salariale stabilite pentru fiecare job în parte;
* Comanda solicitată de fiecare client este realizată de un singur angajat, având job-ul de „bucătar”;
* Un client nu poate solicita două comenzi diferite în aceeași zi;
* Toți clienții care solicită comenzi trebuie să aibă setată o adresă;
* Toate comenzile trebuie să cuprindă cel puțin un preparat;
* Notele clienților vor fi numere naturale cuprinse între 1 și 10;
* Inițial, valoarea comenzii va fi setată NULL, urmând ca mai apoi să fie calculată automat în funcție de prețul și cantitatea preparatelor pe care le cuprinde.

1. Descrierea entităților, incluzând precizarea cheii primare.

Entitățile modelului sunt următoarele:

* RESTAURANT – entitate necesară deoarece modelăm un lanț de restaurante, nu un singur restaurant. Vom reține informații despre fiecare dintre restaurantele noastre. Cheia primară: cod\_restaurant;
* ANGAJAT – persoană fizică, angajată în restaurant; conține detalii despre fiecare angajat din lanțul nostru de restaurante. Cheia primară: cod\_angajat;
* JOB – numele job-ului, cât și limitele salariale stabilite pentru fiecare job în parte. Cheia primară: cod\_job;
* ISTORIC\_JOB – reprezintă istoricul job-urilor pe care le-a avut fiecare angajat. Fiind entitate dependentă, cheia primară include cheia primară a sursei (cod\_angajat) și atributul caracteristic entității (data\_start);
* ADRESA – entitate care identifică locația corespunzătoare fiecărui restaurant și fiecărui client care comandă la unul dintre restaurantele noastre. Cheia primară: cod\_adresa;
* PROGRAM – reprezintă programul de lucru pentru fiecare restaurant. Cheia primară: cod\_program;
* EVENIMENT – entitatea reține detalii despre evenimentele ce vor avea loc în restaurante. Cheia primară: cod\_eveniment;
* TIP\_EVENIMENT – ajută la specializarea fiecărui eveniment (nuntă, botez, majorat etc). Cheia primară: cod\_tip;
* FACILITATE – conține detalii despre facilitățile incluse în funcție de tipul de eveniment. Aceasta este entitate dependentă, iar cheia primară este formată din cheia primară a sursei (cod\_tip) și din atributul caracteristic (cod\_facilitate);
* CLIENT – persoană fizică care solictă o comandă sau face o rezervare în restaurantele noastre. Cheia primară: cod\_client;
* REZERVARE – entitate unde vom reține informațiile despre fiecare rezervare făcută de un client pentru un anumit eveniment. Cheia primară: cod\_rezervare;
* COMANDA – aceasta este formată din detaliile unei comenzi. Cheia primară: cod\_comanda;
* ISTORIC\_COMANDA – reprezintă istoricul comenzilor pe care le-a solicitat fiecare client. Entitatea este dependentă, cheia primară fiind formată din cheia primară a sursei (cod\_client) și din atributul caracteristic (data\_plasare);
* PREPARAT – este reținut fiecare preparat de care dispune lanțul nostru de restaurante. Cheia primară: cod\_preparat.

1. Descrierea relațiilor, incluzând precizarea cardinalității acestora.

*Poate* (cardinalitate maximă) *Trebuie* (cardinalitate minimă)

* Relația RESTAURANT\_realizează\_EVENIMENT:

Într-un restaurant se pot realiza mai multe evenimente (relație dintre entitățile RESTAURANT și EVENIMENT).

Cadinalitatea maximă: many-many (M:M)

* Câte restaurante pot realiza evenimente? Multe!
* Câte evenimente se pot realiza într-un restaurant? Multe!

Cardinalitatea minima: zero-zero (0:0)

* Câte restaurante trebuie să realizeze evenimente? Zero!
* Câte evenimente trebuie să se realizeze într-un restaurant? Zero!
* Relația EVENIMENT\_are\_TIP\_EVENIMENT:

Un eveniment corespunde unui anumit tip: nuntă, botez, majorat etc (relație ce leagă entitățile EVENIMENT și TIP\_EVENIMENT).

Cardinalitatea maximă: many-one (M:1)

* Câte evenimente pot avea un tip? Multe!
* Câte tipuri poate avea un eveniment? Cel mult unul!

Cardinalitatea minimă: zero-zero (0:0)

* Câte evenimente trebuie să aibă un tip? Zero!
* Câte tipuri trebuie să aibă un eveniment? Zero!
* Relația TIP\_EVENIMENT\_oferă\_FACILITATE:

În funcție de tipul de eveniment, restaurantul oferă anumite facilități (relație dintre entitățile TIP\_EVENIMENT și FACILITATE).

Cardinalitatea maximă: many-one (M:1)

* Câte facilități se pot oferi în funcție de tip? Multe!
* În funcție de câte tipuri se pot oferi facilități? Cel mult unul!

Cardinalitatea minimă: zero-one (0:1)

* Câte facilități trebuie oferite în funcție de tip? Zero!
* În funcție de câte tipuri trebuie să se ofere facilități? Cel puțin unul!
* Relația RESTAURANT\_are\_PROGRAM:

Fiecărui restaurant îi corespunde un program de lucru (relație ce leagă entitățile RESTAURANT și PROGRAM).

Cardinalitatea maximă: many-one (M:1)

* Câte restaurante pot avea program? Multe!
* Câte programe poate avea un restaurant? Cel mult unul!

Cardinalitatea minimă: zero-zero (0:0)

* Câte restaurante trebuie să aibă un program? Zero!
* Câte programe trebuie să aibă un restaurant? Zero!
* Relația RESTAURANT\_are\_ADRESA:

Fiecărui restaurant îi corespunde o adresă (relație dintre entitățile RESTAURANT și ADRESA).

Cardinalitatea maximă: many-one (M:1)

* Câte restaurante pot avea adresă? Multe!
* Câte adrese poate avea un restaurant? Cel mult una!

Cardinalitatea minimă: zero-zero (0:0)

* Câte restaurante trebuie să aibă adresă? Zero!
* Câte adrese trebuie să aibă un restaurant? Zero!
* Relația ANGAJAT\_lucrează\_în\_RESTAURANT

Fiecare angajat lucrează într-un anumit restaurant (relație ce leagă entitățile ANGAJAT și RESTAURANT).

Cardinalitate maximă: many-one (M:1)

* Câți angajați pot lucra într-un restaurant? Mulți!
* În câte restaurante poate lucra un angajat? În cel mult unul!

Cardinalitatea minimă: zero-zero (0:0)

* Câți angajați trebuie să lucreze într-un restaurant? Zero!
* În câte restaurante trebuie să lucreze un angajat? Zero!
* Relația ANGAJAT\_conduce\_ANGAJAT:

Orice angajat este condus de către un alt angajat, cu excepția șefului restaurantului (relație dintre entitățile ANGAJAT și ANGAJAT).

Cardinalitatea maximă: many-one (M:1)

* Câți angajați poate conduce un alt angajat? Mulți!
* Câți angajați pot conduce un alt angajat? Cel mult unul!

Cardinalitatea minimă: zero-zero (0:0)

* Câți angajați trebuie să aibă în subordine un alt angajat? Zero!
* Câți angajați trebuie să conducă un alt angajat? Zero!
* Relația ANGAJAT\_are\_JOB:

Un angajat al restaurantului are un anumit job (relație ce leagă entitățile ANGAJAT și JOB).

Cardinalitatea maximă: many-one (M:1)

* Câți angajați pot avea un job? Mulți!
* Câte job-uri poate avea un angajat? Cel mult unul!

Cardinalitaea minimă: many-one (M:1)

* Câți angajați trebuie să aibă un job? Mulți!
* Câte job-uri trebuie să aibă un angajat? Cel puțin unul!
* Relația JOB\_are\_ISTORIC\_JOB:

Fiecare istoric are asociat un anumit job (relație dintre entitățile JOB și ISTORIC\_JOB).

Cardinalitatea maximă: one-many (1:M)

* Câte job-uri pot avea un anumit istoric? Cel mult unul!
* Câte istorice poate avea un job? Multe!

Cardinalitatea minimă: one-zero (1:0)

* Câte job-uri trebuie să aibă un anumit istoric? Cel puțin unul!
* Câte istorice trebuie să aibă un job? Zero!
* Relația ANGAJAT\_are\_ISTORIC\_JOB:

Fiecare istoric are asociat un anumit angajat (relație ce leagă entitățile ANGAJAT și ISTORIC\_JOB).

Cardinalitatea maximă: one-many (1:M)

* Câți angajați pot avea un anumit istoric? Cel mult unul!
* Câte istorice poate avea un angajat? Multe!

Cardinalitatea minimă: one-zero (1:0)

* Câți angajați trebuie să aibă un anumit istoric? Cel puțin unul!
* Câte istorice trebuie să aibă un angajat? Zero!
* Relația COMANDA\_plasată\_în\_RESTAURANT:

Orice comandă plasată de către un client se realizează în cadrul unui anumit restaurant (relație dintre entitățile COMANDA și RESTAURANT).

Cardinalitate maximă: many-one (M:1)

* Câte comenzi pot fi plasate într-un restaurant? Multe!
* În câte restaurante poate fi plasată o comandă? În cel mult unul!

Cardinalitate minimă: zero-zero (0:0)

* Câte comenzi trebuie plasate într-un restaurant? Zero!
* În câte restaurante trebuie plasată o comandă? Zero!
* Relația ANGAJAT\_realizează\_COMANDA:

Fiecare comandă plasată este realizată de către un singur angajat (relație ce leagă entitățile ANGAJAT și COMANDA).

Cardinalitatea maximă: one-many (1:M)

* Câți angajați pot realiza o comandă? Cel mult unul!
* Câte comenzi poate realiza un angajat? Multe!

Cardinalitatea minimă: zero-zero (0:0)

* Câți angajați trebuie să realizeze o comandă? Zero!
* Câte comenzi trebuie realizate de către un angajat? Zero!
* Relația CLIENT\_face\_REZERVARE\_la\_un\_EVENIMENT\_dintr-un\_RESTAURANT:

Aceasta este o relație de ordin superior (4) deoarece folosește patru entități pentru a răspunde cât mai corect și eficient întrebărilor ce pot să apară în legătură cu rezervările clienților (relație dintre entitățile CLIENT, REZERVARE, EVENIMENT și RESTAURANT).

Cardinalitatea maximă: many-many-many-many (M:M:M:M)

* Câți clienți pot face rezervare pentru un eveniment dintr-un anumit restaurant? Mulți!
* La câte evenimente dintr-un anumit restaurant poate face rezervare un client? La multe!
* Câte rezervări la un eveniment dintr-un anumit restaurant poate face un client? Multe!
* La câte restaurante poate face rezervare un client pentru un eveniment? La multe!

Cardinalitatea minimă: zero-zero-zero-zero (0:0:0:0)

* Câți clienți trebuie să facă rezervare pentru un eveniment dintr-un anumit restaurant? Zero!
* La câte evenimente dintr-un anumit restaurant trebuie să facă rezervare un client? Zero!
* Câte rezervări la un eveniment dintr-un anumit restaurant trebuie să facă un client? Zero!
* La câte restaurante trebuie să facă rezervare un client pentru un eveniment? Zero!
* Relația CLIENT\_solicită\_COMANDA:

Clienții pot solicita comenzi (relație ce leagă entitățile CLIENT și COMANDA).

Cardinalitatea maximă: many-one (M:1)

* Câți clienți pot solicita comenzi? Mulți!
* Câte comenzi poate solicita un client? Cel mult una!

Cardinalitatea minimă: zero-zero (0:0)

* Câți clienți trebuie să solicite comenzi? Zero!
* Câte comenzi trebuie să fie solicitate de un client? Zero!
* Relația CLIENT\_are\_ISTORIC\_COMANDA

Fiecare istoric are asociat un anumit client (relație dintre entitățile CLIENT și ISTORIC\_COMANDA).

Cardinalitatea maximă: one-many (1:M)

* Câți clienți pot avea un anumit istoric? Cel mult unul!
* Câte istorice poate avea un client? Multe!

Cardinalitatea minimă: one-zero (1:0)

* Câți clienți trebuie să aibă un anumit istoric? Cel puțin unul!
* Câte istorice trebuie să aibă un client? Zero!
* Relația COMANDA\_are\_ISTORIC\_COMANDA

Fiecare istoric are asociată o anumită comandă (relație ce leagă entitățile COMANDA și ISTORIC\_COMANDA).

Cardinalitatea maximă: one-many (1:M)

* Câte comenzi pot avea un anumit istoric? Cel mult una!
* Câte istorice poate avea o comandă? Multe!

Cardinalitatea minimă: one-zero (1:0)

* Câte comenzi trebuie să aibă un anumit istoric? Cel puțin una!
* Câte istorice trebuie să aibă o comandă? Zero!
* Relația COMANDA\_cuprinde\_PREPARAT

Fiecare comandă este formată din anumite preparate în diverse cantități (relație dintre entitățile COMANDA și PREPARAT).

Cardinalitatea maximă: many-many (M:M)

* Câte comenzi pot cuprinde preparate? Multe!
* Câte preparate poate cuprinde o comandă? Multe!

Cardinalitatea minimă: many-one (M:1)

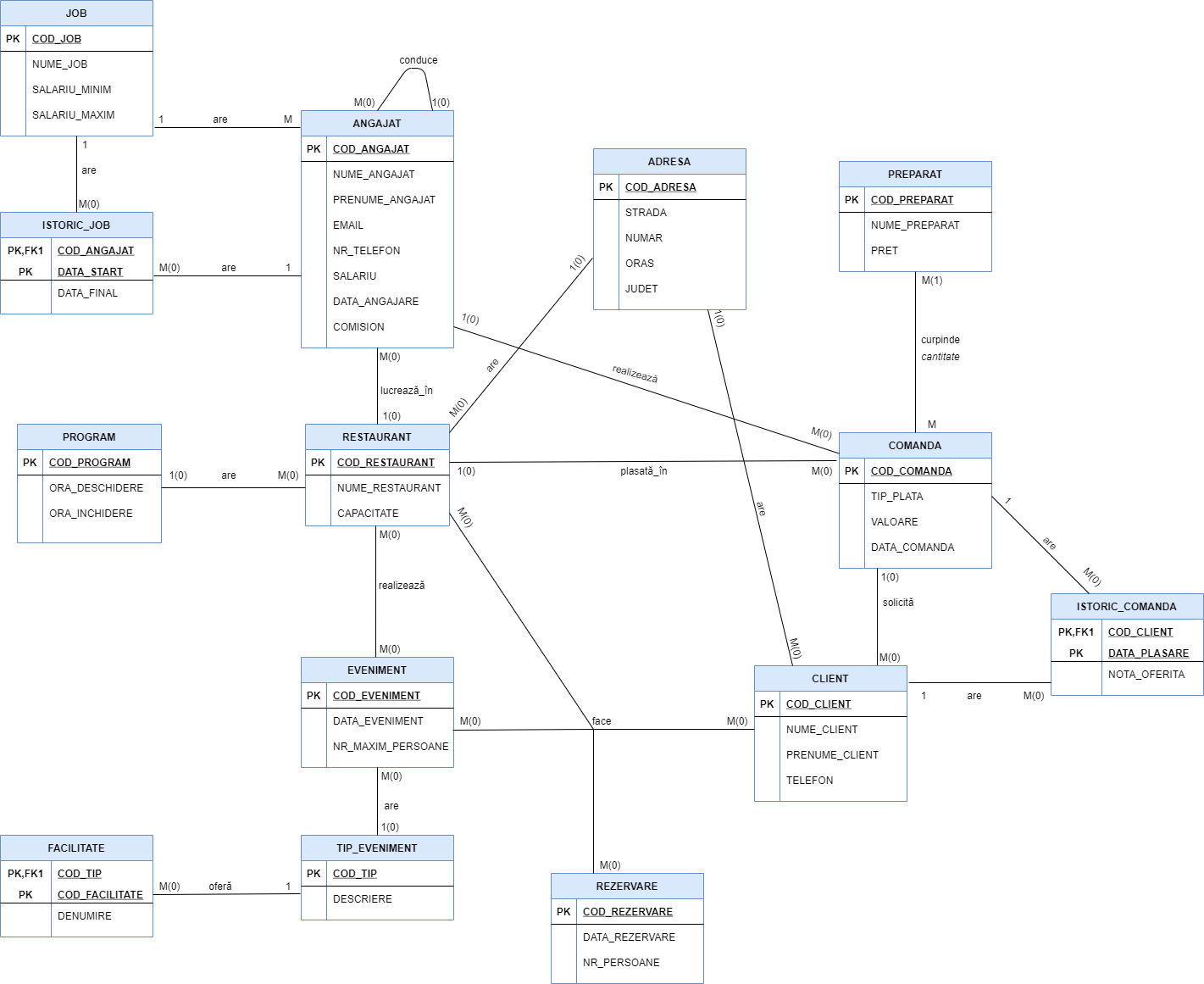
* Câte comenzi trebuie să cuprinde preparate? Multe!
* Câte preparate trebuie să cuprindă o comandă? Cel puțin unul!

1. Descrierea atributelor, incluzând tipul de date și eventualele constrângeri, valori implicite, valori posibile ale atributelor.

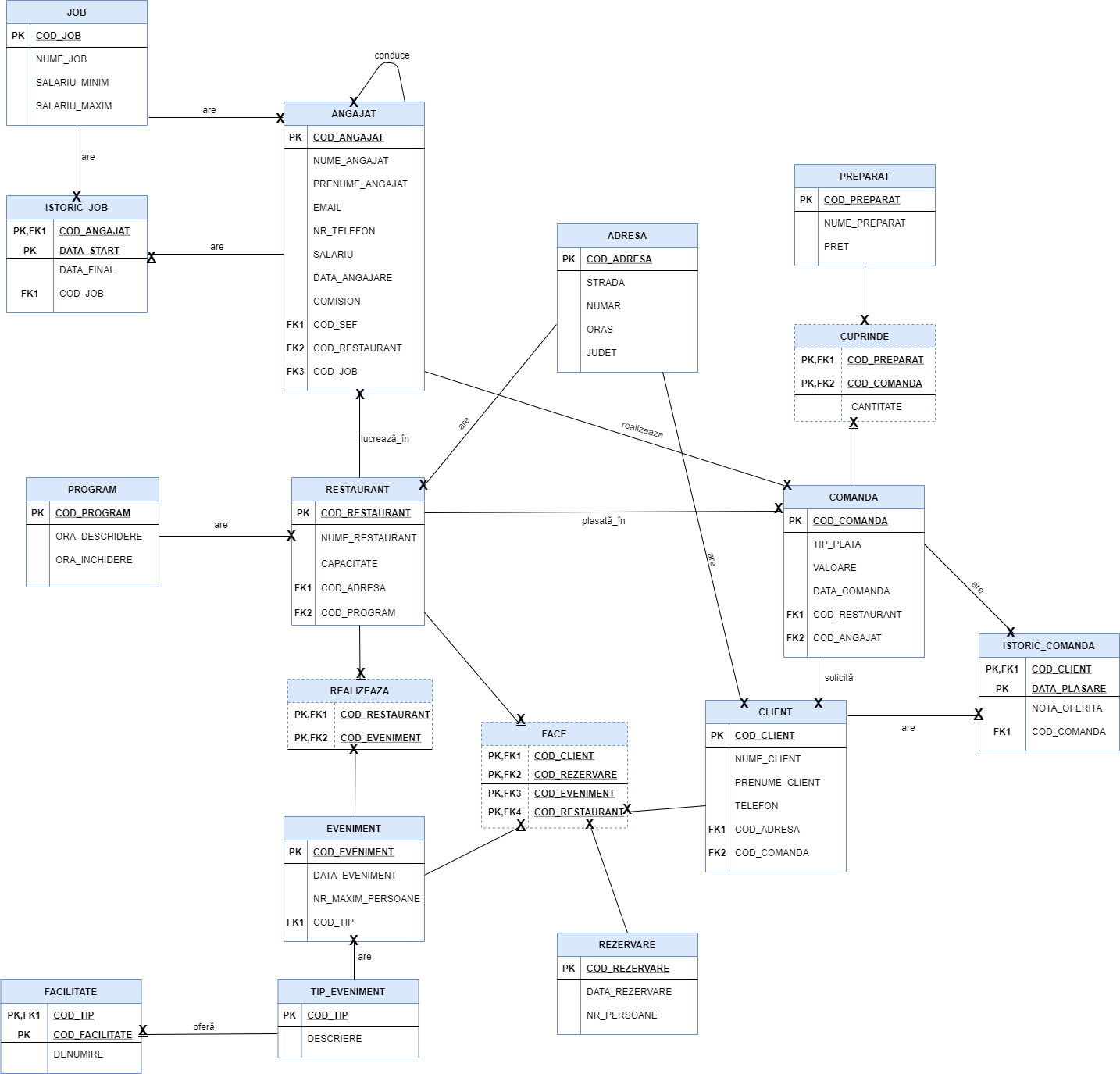
|  |  |
| --- | --- |
| **ENTITĂȚI** | **ATRIBUTE** |
| RESTAURANT | * cod\_restaurant: constrângere de cheie   primară, este automat not null și unique, nu are valori default, iar tipul de date este number(4);   * nume\_restaurant: reține numele   fiecărui restaurant, nu poate fi null (constrângere de tip check), tipul de date fiind varchar2(30);   * capacitate: numărul de persoane   disponibil pentru fiecare restaurant; poate fi null; tipul de date este number(4);   * cod\_program: cheie externă ce   referențiază tabela PROGRAM, cu ajutorul căreia aflăm programul fiecărui restaurant, poate fi null; tip de date: number(2);   * cod\_adresa: cheie externă ce   referențiază tabela ADRESA, cu ajutorul căreia aflăm adresa fiecărui restaurant, poate fi null; tip de date: number(4). |
| PROGRAM | * cod\_program: constrângere de cheie   primară, este automat not null și unique, nu are valori default, iar tipul de date este number(2);   * ora\_deschidere: reprezintă ora la care se   deschide restaurantul, poate fi null, iar tipul de date este varchar2(6) – (ora sub forma „08.00”);   * ora\_inchidere: reprezintă ora la care se   închide restaurantul, poate fi null, iar tipul de date este varchar2(6) – (ora sub forma „21.00”). Se va respecta condiția ca ora\_inchidere să fie mai mare decât ora\_deschidere (check). |
| EVENIMENT | * cod\_eveniment: constrângere de cheie   primară, este automat not null și unique, nu are valori default, iar tipul de date este number(4);   * data\_eveniment: reprezintă data la care   are loc evenimentul pentru care s-a făcut rezervare; poate fi null; tip de date: date.   * nr\_maxim\_persoane: reține numărul de   persoane din cadrul unui eveniment, poate fi null; tip de date: number(4);   * cod\_tip: cheie externă ce   referențiază tabela TIP\_EVENIMENT, cu ajutorul căreia știm ce tip are un anumit eveniment; poate fi null; tip de date: number(2). |
| TIP\_EVENIMENT | * cod\_tip: constrângere de cheie   primară, este automat not null și unique, nu are valori default, iar tipul de date este number(2);   * descriere: conține tipul evenimentului   (nuntă, botez etc), nu poate fi null (constrângere de tip check); tip de date: varchar2(30). |
| FACILITATE | Fiind entitate dependentă, cheia primară este compusă.   * cod\_tip: constrângere de cheie   primară, este automat not null și unique, constrângere de cheie externă ce referențiază tabela TIP\_EVENIMENT, nu are valori default, iar tipul de date este number(2);   * cod\_facilitate: constrângere de cheie   primară, este automat not null si unique, nu are valori default, iar tipul de date este number(2);   * denumire: atribut ce reține numele   facilității oferite (candy bar, aranjamente florale etc) în funcție de tipul de eveniment. Poate fi null, tipul de date fiind varchar2(50). |
| ADRESA | * cod\_adresa: constrângere de cheie   primară, este automat not null și unique, nu are valori default, iar tipul de date este number(4);   * strada: reține numele străzii, nu poate fi   null (constrângere de tip check), tipul de date este varchar2(30);   * numar: reține numărul adresei, nu poate   fi null (constrângere de tip check), tipul de date este number(3);   * oras: reține numele orașului, nu poate fi   null (constrângere de tip check), tipul de date este varchar2(30);   * județ: reține numele județului, poate fi   null, tipul de date este varchar2(30). |
| ANGAJAT | * cod\_angajat: constrângere de cheie   primară, este automat not null și unique, nu are valori default, iar tipul de date este number(4);   * nume\_angajat: reprezintă numele   angajatului; nu poate fi null; tip de date: varchar2(25);   * prenume\_angajat: reprezintă prenumele   angajatului; acesta poate fi null; tip  de date: varchar2(25). Nume\_angajat împreuna cu prenume\_angajat trebuie să aibă combinație unică;   * email: reține email-ul angajatului; acest   atribut are constrângere de tip check (not null) și de unicitate; tip de date: varchar2(25);   * nr\_telefon: reține numărul de telefon,   trebuie să fie unic și diferit de null; tip de date: varchar2(15);   * salariu: reține salariul angajatului; acesta   poate fi null, în schimb avem o constrângere de tip check ( trebuie sa fie mai mare decât 0); tip de date: number(8,2);   * data\_angajare: reține data la care   salariatul a fost angajat, este by default egală cu data curentă; tip de date: date;   * comision: reprezintă comisionul   angajatului; poate fi null, tipul de date fiind  number(2,2);   * cod\_restaurant: cheie externă ce   referențiază tabela RESTAURANT, cu ajutorul căreia știm în ce restaurant lucrează fiecare angajat; poate fi null; tip de date: number(4);   * cod\_sef: cheie externă ce referențiază   tabela ANGAJAT, reține codul șefului fiecărui angajat cu excepția angajatului ce este administratorul restaurantului; poate fi null; tip de date: number(4);   * cod\_job: cheie externă ce referențiază   tabela JOB, astfel știind ce job are fiecare angajat; nu poate fi null; tip de date: number(4). |
| JOB | * cod\_job: cheie primară, automat not null   și unique; tip de date: number(4);   * nume\_job: reține numele job-ului, nu   poate fi null; tip de date: varchar2(35);   * salariu\_minim: reprezintă limita salarială   inferioară, poate fi null; tip de date: number(6);   * salariu\_maxim: reprezintă limita salarială   superioară, poate fi null; tip de date: number(6). |
| ISTORIC\_JOB | Fiind entitate dependentă, cheia primară este compusă.   * cod\_angajat: constrângere de cheie   primară, este automat not null și unique, constrângere de cheie externă ce referențiază tabela ANGAJAT, nu are valori default, iar tipul de date este number(4);   * data\_start: constrângere de cheie   primară, este automat not null și unique, este by default data curentă, iar tipul de date este date;   * data\_final: reprezintă data când angajatul   a încetat să mai lucreze pe acel post, poate fi null; tip de date: date;   * cod\_job: cheie externă ce referențiază   tabela JOB, astfel știind ce job a avut fiecare angajat pe parcursul timpului; nu poate fi null (constrângere check); tip de date: number(4). |
| CLIENT | * cod\_client: constrângere de cheie   primară, not null și unique; tip de date: number(4);   * nume\_client: reține numele clientului,   nu poate fi null; tip de date: varchar2(25);   * prenume\_client: reține prenumele   clientului, acesta poate fi null; tip de date: varchar2(25). Nume\_client împreună cu prenume\_client trebuie să aibă combinație unică.   * telefon: numărul de telefon trebuie să   fie unic și diferit de null (check); tip de date: varchar2(15);   * cod\_adresa: cheie externă ce   referențiază tabela ADRESA, cu ajutorul căreia aflăm adresa fiecărui client, poate fi null; tip de date: number(4);   * cod\_comanda: cheie externă ce   referențiază tabela COMANDA, cu ajutorul căreia aflăm diverse detalii despre comanda, poate fi null; tip de date: number(4). |
| COMANDA | * cod\_comanda: constrângere de cheie   primară, not null și unique; tip de date: number(4);   * tip\_plata: reține informații despre plata   comenzii (card/cash), poate fi null; tip de date: varchar2(20);   * valoare: reprezintă valoarea comenzii,   care inițial este null peste tot, urmând să fie actualizată folosind tabela PREPARATE și prețul fiecărui preparat. Tip de date: number(8,2);   * data\_comanda: reține data când a fost   plasată comanda, poate fi null, nu avem valori default; tip de date: date;   * cod\_angajat: cheie externă ce   referențiază tabela ANGAJAT, astfel știind ce angajat a realizat comanda, poate fi null; tip de date: number(4);   * cod\_restaurant: cheie externă ce   referențiază tabela RESTAURANT, astfel știind în ce restaurant s-a plasat și s-a realizat comanda, poate fi null; tip de date: number(4). |
| ISTORIC\_COMANDA | Fiind entitate dependentă, cheia primară este compusă.   * cod\_client: constrângere de cheie   primară, este automat not null și unique, constrângere de cheie externă ce referențiază tabela CLIENT, nu are valori default, iar tipul de date este number(4);   * data\_plasare: constrângere de cheie   primară, este automat not null si unique, este by default data curentă, iar tipul de date este date;   * nota\_oferita: reprezintă nota pe care   clientul a oferit-o pentru comanda plasată (preparate, livrare etc). Poate fi null, tipul de date fiind number(2);   * cod\_comanda: cheie externă ce   referențiază tabela COMANDA, astfel știind istoricul comenzilor pentru fiecare client; nu poate fi null (check), iar tipul de date este number(4). |
| PREPARAT | * cod\_preparat: constrângere de cheie   primară, not null și unique; tip de date: number(4);   * nume\_preparat: reține numele   preparatului, nu poate fi null; tip de date: varchar2(25);   * pret: reține prețul preparatului,   nu poate fi null deoarece avem nevoie de pretul preparatelor pentru a totaliza valoarea comenzii din tabela COMANDA; tip de date: number(8,2). |
| REZERVARE | * cod\_rezervare: constrângere de cheie   primară, not null și unique; tip de date: number(6);   * data\_rezervare: reprezintă data la care   clientul a făcut o rezervare pentru un eveniment, este by default data curentă în caz că nu se specifică. Tip de date: date;   * nr\_persoane: reține numărul de persoane   pentru care se face rezervarea; poate fi null; tip de date: number(3). |

În plus, pe relația COMANDA\_cuprinde\_PREPARAT avem atributul cantitate, care reprezintă cantitatea preparatului dintr-o anumită comandă, este not null, tipul de date fiind number(2).

1. Realizarea diagramei entitate-relație corespunzătoare descrierii de la punctele 3-5.



1. Realizarea diagramei conceptuale corespunzătoare diagramei entitate-relație proiectate la punctul 6.



1. Enumerarea schemelor relaționale corespunzătoare diagramei conceptuale proiectate la punctul 7.

Schemele relaționale corespunzătoare diagramei conceptuale sunt următoarele:

* **RESTAURANT**(cod\_restaurant#, nume\_restaurant, capacitate, cod\_adresa, cod\_program);
* **EVENIMENT**(cod\_eveniment#, data\_eveniment, nr\_maxim\_persoane, cod\_tip);
* **REALIZEAZA**(cod\_restaurant#, cod\_eveniment#);
* **TIP\_EVENIMENT**(cod\_tip#, descriere);
* **FACILITATE**(cod\_tip#, cod\_facilitate#, denumire);
* **PROGRAM**(cod\_program#, ora\_deschidere, ora\_inchidere);
* **ADRESA**(cod\_adresa#, strada, numar, oras, judet);
* **ANGAJAT**(cod\_angajat#, nume\_angajat, prenume\_angajat, email, nr\_telefon, salariu, data\_angajare, comision, cod\_sef, cod\_restaurant, cod\_job);
* **JOB**(cod\_job#, nume\_job, salariu\_minim, salariu\_maxim);
* **ISTORIC\_JOB**(cod\_angajat#, data\_start#, data\_final, cod\_job);
* **CLIENT**(cod\_client#, nume\_client, prenume\_client, telefon, cod\_adresa, cod\_comanda);
* **REZERVARE**(cod\_rezervare#, data\_rezervare, nr\_persoane);
* **FACE**(cod\_client#, cod\_rezervare#, cod\_eveniment#, cod\_restaurant#);
* **COMANDA**(cod\_comanda#, tip\_plata, valoare, data\_comanda, cod\_restaurant, cod\_angajat);
* **PREPARAT**(cod\_preparat#, nume\_preparat, pret);
* **CUPRINDE**(cod\_comanda#, cod\_preparat#, cantitate);
* **ISTORIC\_COMANDA**(cod\_client#, data\_plasare#, nota\_oferita, cod\_comanda).

1. Realizarea normalizării până la forma normală 3 (FN1-FN3).

* **FN1**

O relație este în FN1 dacă fiecărui atribut care o compune îi corespunde o

valoare indivizibilă.

*Non FN1:*

|  |  |
| --- | --- |
| Cod\_restaurant# | Cod\_eveniment |
| R1 | E1, E2 |
| R2 | E3, E4, E5 |
| R3 | E6, E7 |

După cum putem observa, acest tabel nu se află în FN1 deoarece fiecărui

restaurant îi corespund mai multe evenimente în aceeași înregistrare. Vom rezolva această problemă cu tabelul asociativ dintre entitatea RESTAURANT și EVENIMENT:

|  |  |
| --- | --- |
| Cod\_restaurant# | Cod\_eveniment# |
| R1 | E1 |
| R1 | E2 |
| R2 | E3 |
| R2 | E4 |
| R2 | E5 |
| R3 | E6 |
| R3 | E7 |

* **FN2**

O relație este în FN2 dacă și numai dacă aceasta este în FN1 și fiecare atribut

care nu participă la cheia primară este dependent de întreaga cheie primară.

*Non FN2:*

CUPRINDE:

|  |  |  |  |
| --- | --- | --- | --- |
| Cod\_comanda# | Cod\_preparat# | Nume\_preparat | Cantitate |
| C1 | P1 | Paste\_cu\_ton | 1 |
| C1 | P2 | Pizza\_Capriciosa | 2 |
| C2 | P1 | Paste\_cu\_ton | 2 |
| C3 | P3 | Cartofi\_la\_cuptor | 1 |

Acest tabel nu este în FN2 pentru că nume\_preparat nu depinde de întreaga cheie primară, ci doar de cod\_preparat#. Vom transforma în FN2 folosind următoarele tabele:

CUPRINDE\_1:

|  |  |  |
| --- | --- | --- |
| Cod\_comanda# | Cod\_preparat# | Cantitate |
| C1 | P1 | 1 |
| C1 | P2 | 2 |
| C2 | P1 | 2 |
| C3 | P3 | 1 |

CUPRINDE\_2:

|  |  |
| --- | --- |
| Cod\_preparat# | Nume\_preparat |
| P1 | Paste\_cu\_ton |
| P2 | Pizza\_Capriciosa |
| P3 | Cartofi\_la\_cuptor |

În modelul nostru, CUPRINDE\_1 devine tabelul asociativ dintre PREPARAT și COMANDA, iar CUPRINDE\_2 este de fapt entitatea PREPARAT.

* **FN3**

O relație este în FN3 dacă și numai dacă aceasta este în FN2 și fiecare atribut

care nu participă la o cheie depinde direct de cheia primară.

*Non FN3:*

|  |  |  |  |
| --- | --- | --- | --- |
| Cod\_angajat# | Nume\_angajat | Cod\_job | Nume\_job |
| A1 | Popescu | J1 | Bucătar |
| A2 | Ion | J2 | Manager |
| A3 | Avramescu | J1 | Bucătar |

Tabelul nu este în FN3 deoarece nume\_job depinde tranzitiv de cheia primară cod\_angajat#: cod\_angajat# cod\_job nume\_job. Transformarea în FN3 este următoarea:

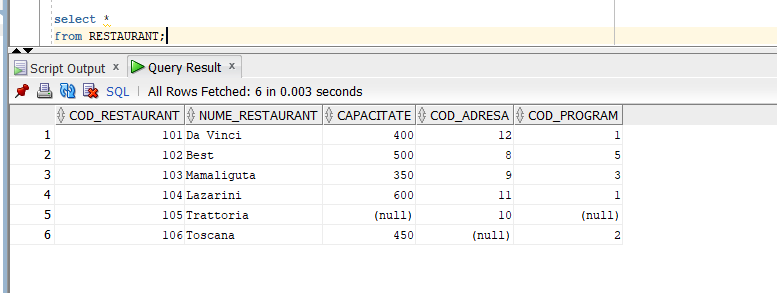
|  |  |  |
| --- | --- | --- |
| Cod\_angajat# | Nume\_angajat | Cod\_job |
| A1 | Popescu | J1 |
| A2 | Ion | J2 |
| A3 | Avramescu | J1 |

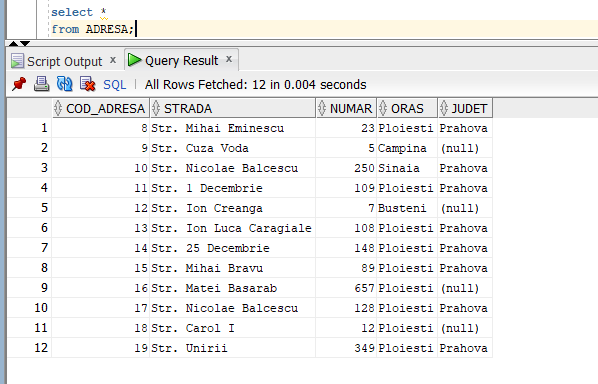
|  |  |
| --- | --- |
| Cod\_job# | Nume\_job |
| J1 | Bucătar |
| J2 | Manager |

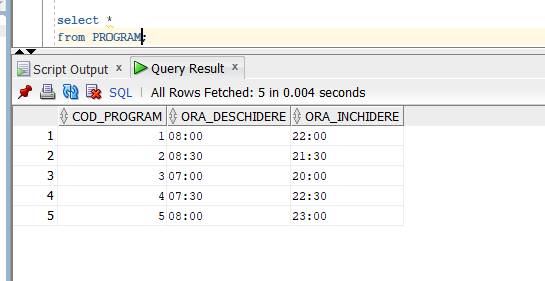
Aceste tabele înglobează relația ANGAJAT\_are\_JOB, cod\_job fiind cheie externă în tabela ANGAJAT și cheie primară în tabela JOB.

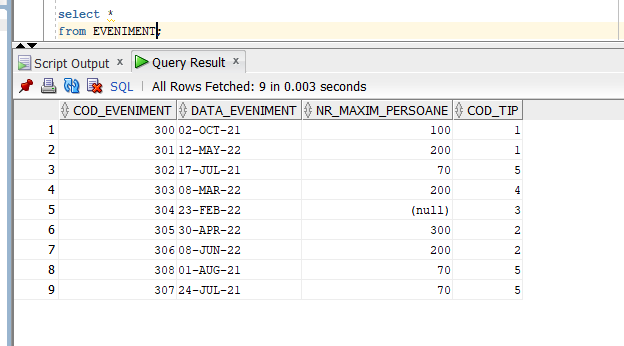
1. Crearea tabelelor în SQL și inserarea de date coerente în fiecare dintre

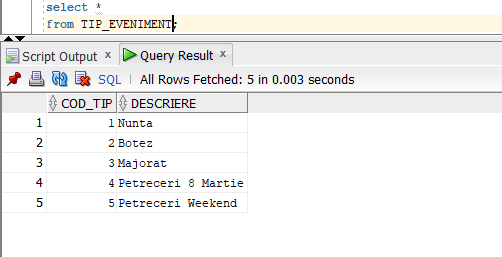
acestea.

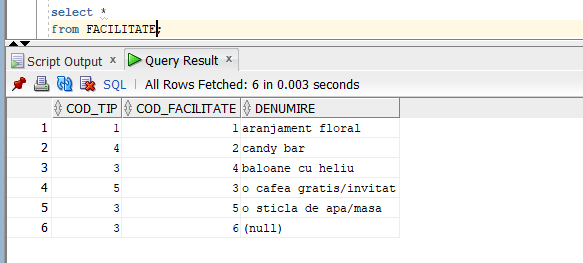


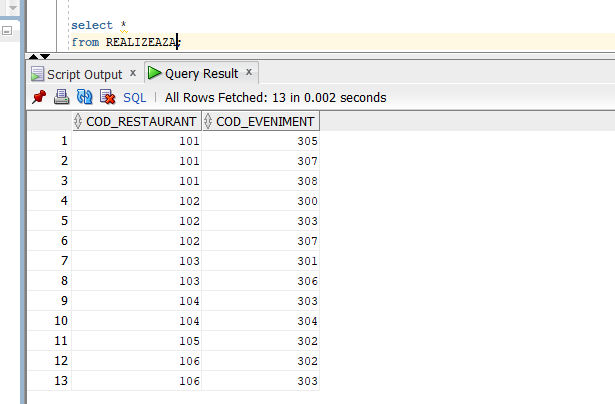


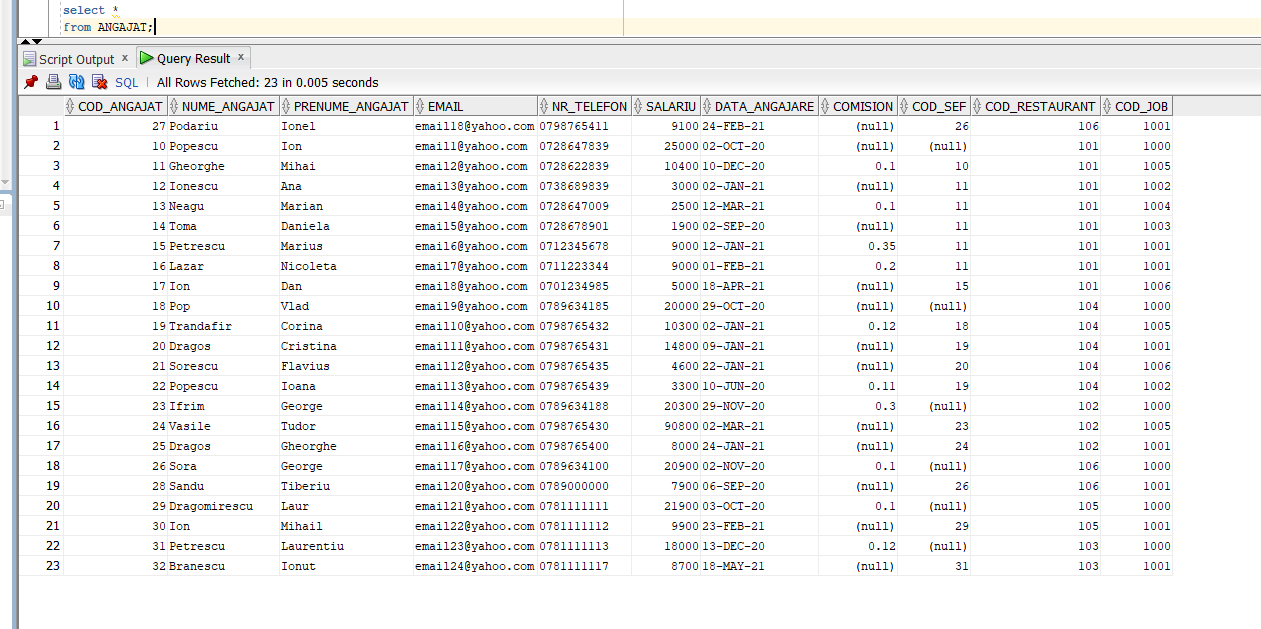


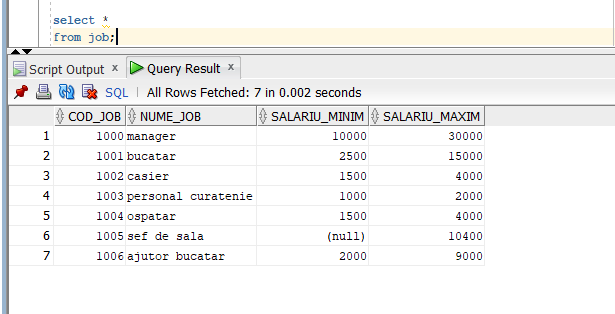


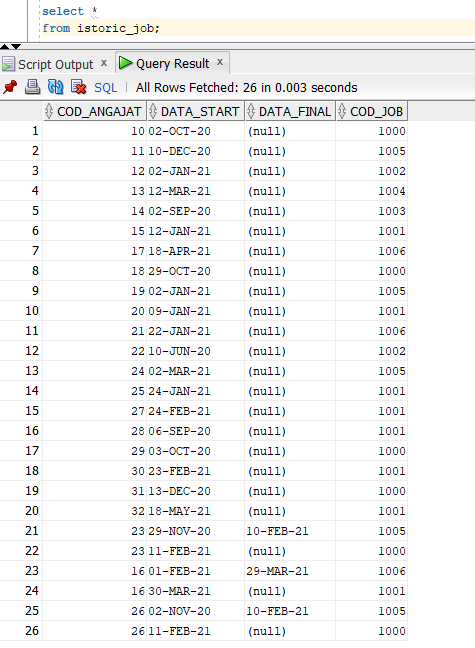


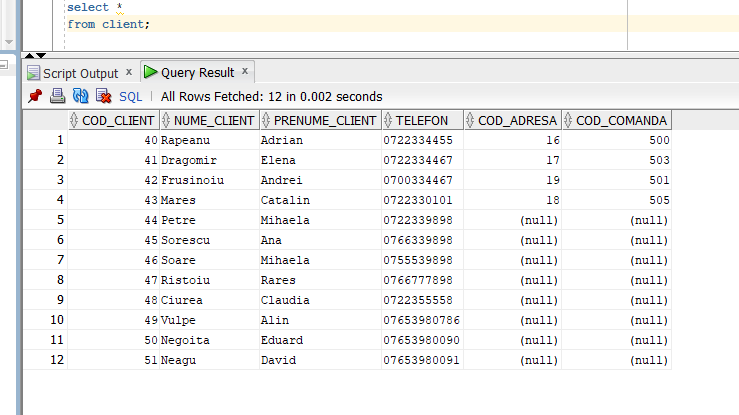


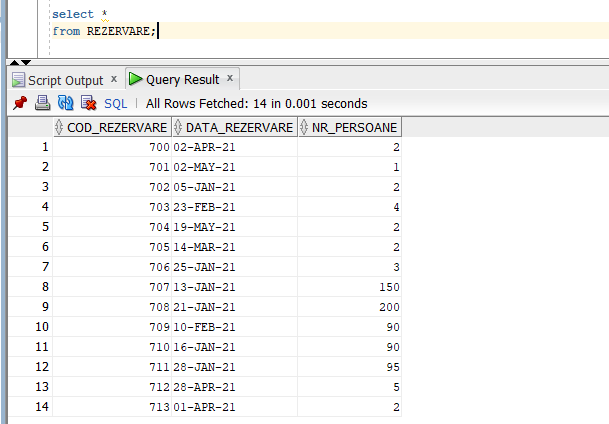


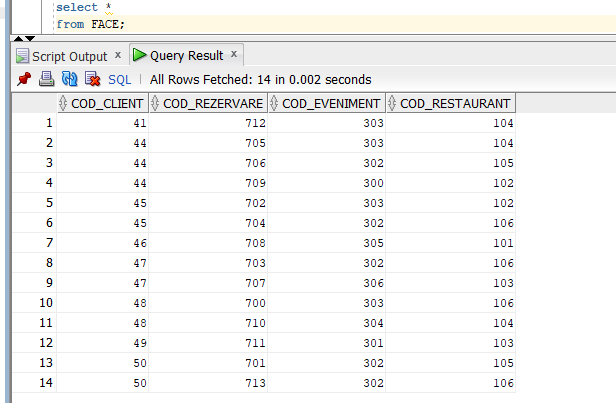


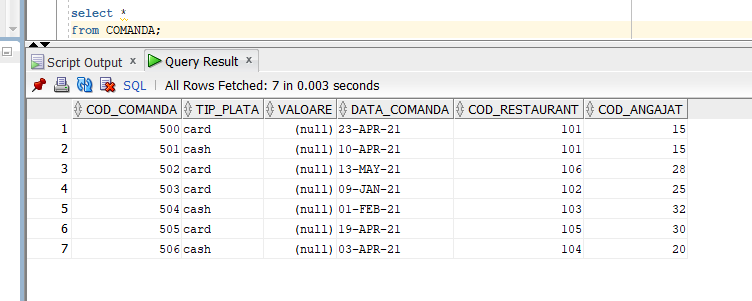


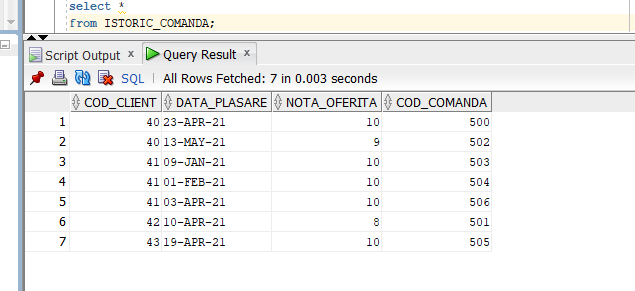


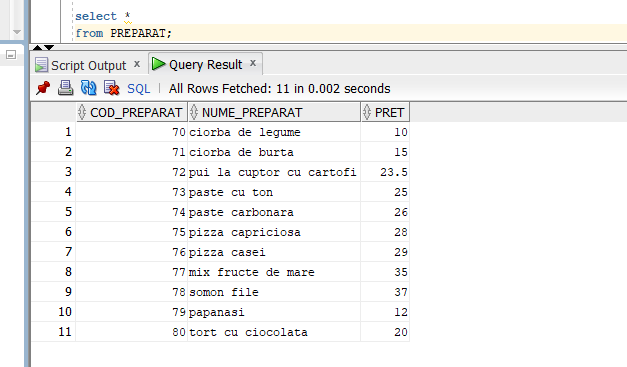


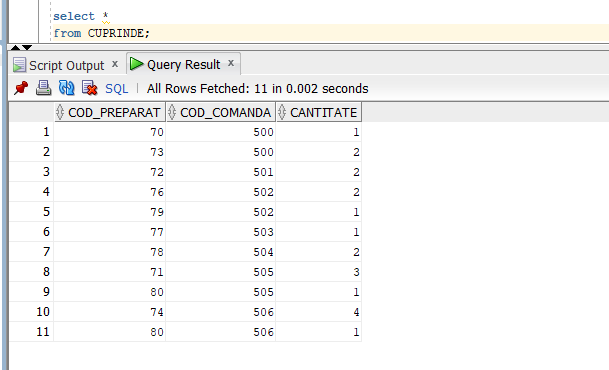








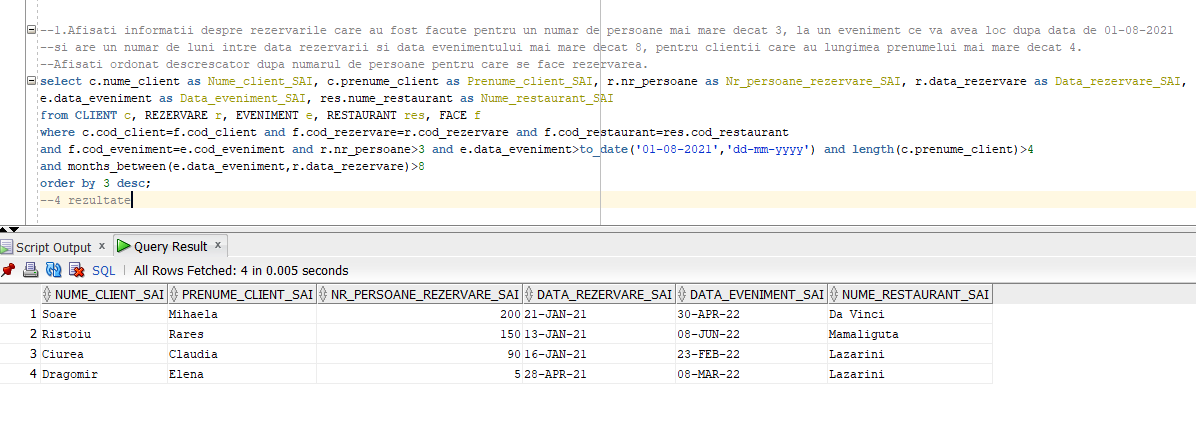




1. Formulați în limbaj natural și implementați 5 cereri SQL complexe ce vor

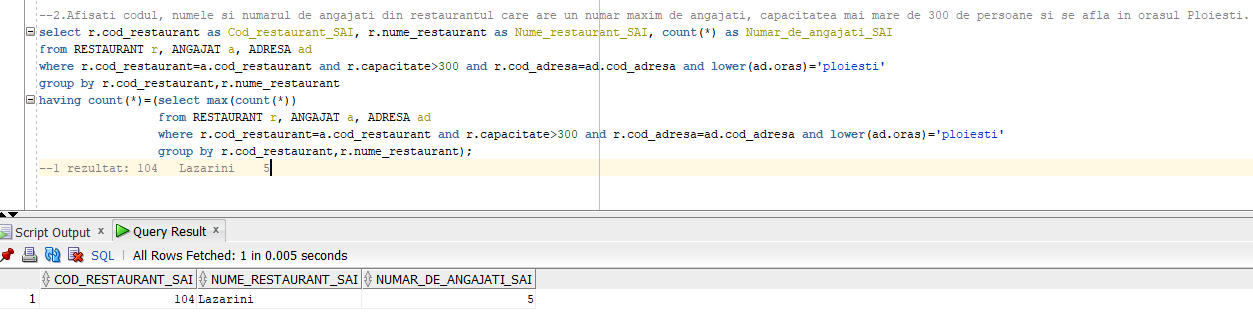
utiliza, în ansamblul lor, următoarele elemente: operație join pe cel puțin 4 tabele; filtrare la nivel de linii; subcereri sincronizate în care intervin cel puțin 3 tabele; subcereri nesincronizate în care intervin cel puțin 3 tabele; grupări de date, funcții grup, filtrare la nivel de grupuri; ordonări; utilizarea a cel puțin 2 funcții pe șiruri de caractere, 2 funcții pe date calendaristice, a funcțiilor NVL și DECODE, a cel puțin unei expresii CASE; utilizarea a cel puțin un bloc de cerere (clauza WITH).

Cererea 1:



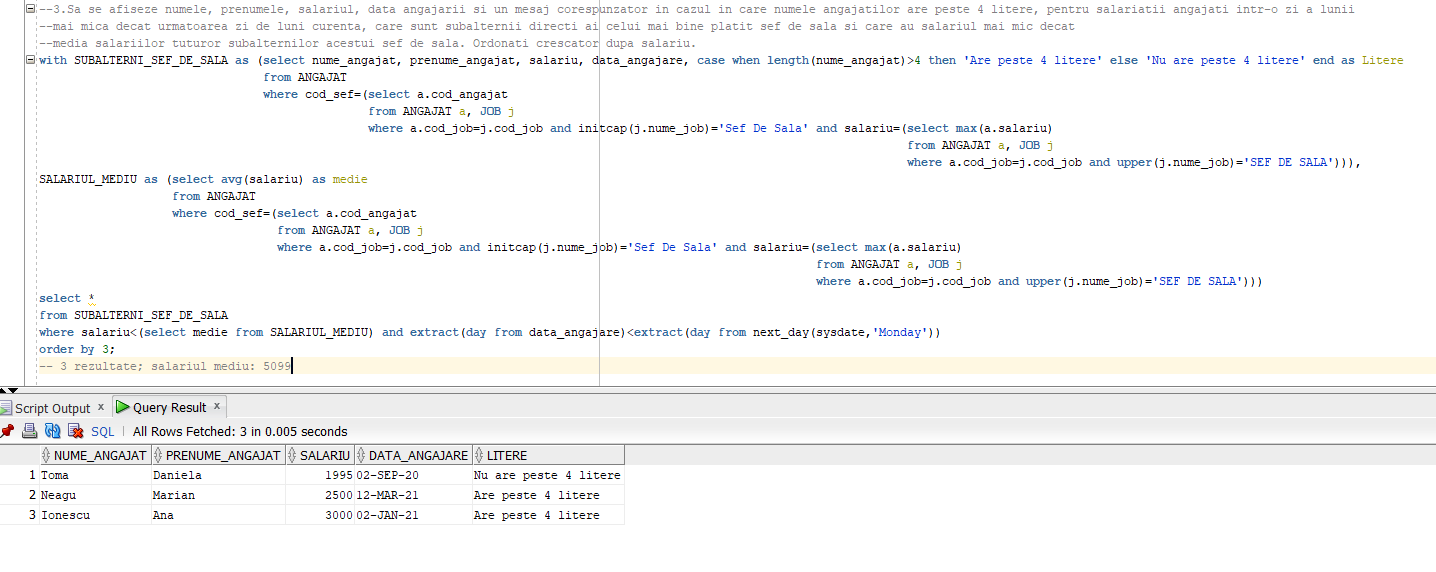
Pentru rezolvarea cerinței am utilizat un join pe tabelele corespunzătoare. Am verificat ca data evenimentului să fie mai mare decât 01-08-2021 și am utilizat funcția months\_between pentru a afla numărul de luni dintre data evenimentului și data rezervării. Pentru determinarea clienților cu lungimea prenumelui mai mare decât 4 am folosit funcția length.

Cererea 2:



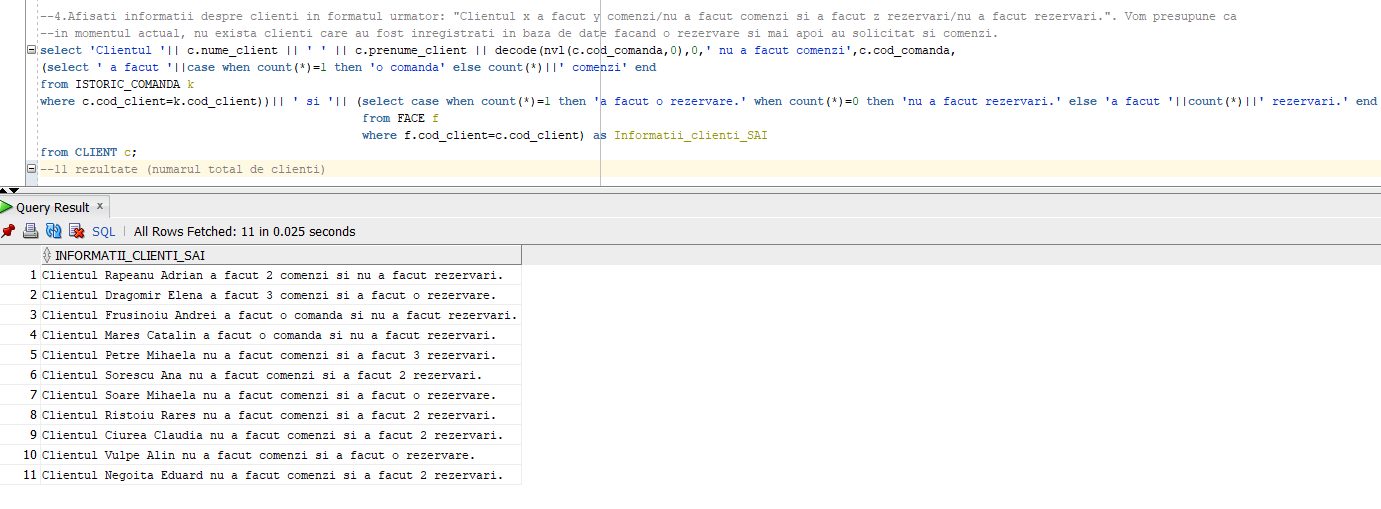
Pentru a afla numărul maxim de angajați am folosit o subcerere necorelată în care am adăugat condițiile necesare conform cerinței (capacitatea și orașul adresei). Funcțiile grup utilizate: MAX (întoarce maximul) și COUNT (returneaza numărul de angajați). S-a grupat după codul și numele restaurantului

Cererea 3:



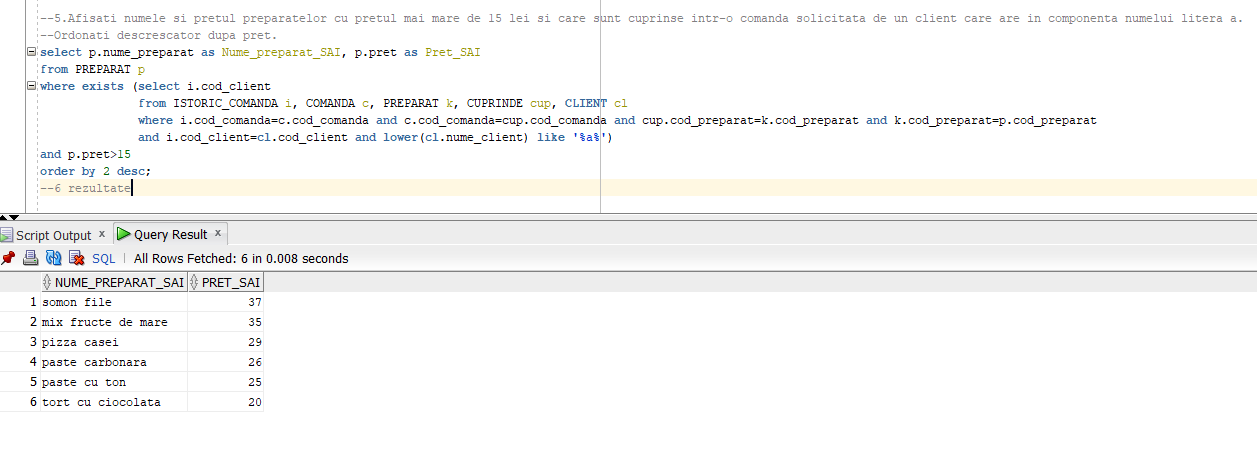
Tabelul SUBALTERNI\_SEF\_DE\_SALA va reține toți angajații care sunt subalternii direcți ai celui mai bine plătit șef de sală. Pentru a stabili dacă șeful respectiv are salariul maxim am folosit o subcerere necorelată, alături de funcția grup MAX. Tabelul SALARIUL\_MEDIU va reține media salariilor tuturor angajaților care sunt subalternii acestui șef. Media se va calculca utilizând funcția grup AVG. Vom selecta din primul tabel doar acei angajați care au salariul mai mic decât media și au ziua din data de angajare mai mică decât următoarea zi de luni curentă. Pentru a stabili acest detaliu, am folosit next\_day(sysdaye, ‘Monday’), unde sysdate returnează data curentă. Cu ajutorul clauzei CASE, am afișat mesaj corespunzător în legătură cu lungimea numelui angajatului.

Cererea 4:



Pentru a afișa sub formatul dorit în cerință, am folosit operatorul „||” care concatenează coloanele, alături de string-uri. Am folosit funcțiile NVL și DECODE în felul următor: NVL(cod\_comanda,0) – dacă codul comenzii este NULL, se va returna 0, altfel, se va returna codul; DECODE – dacă codul comenzii este NULL, se va afișa mesaj corespunzător, altfel, cu ajutorul unei subcereri corelate am numărat câte comenzi a făcut fiecare client. În această subcerere, am folosit clauza CASE pentru a afișa corespunzător mesajul „o comandă/nr comenzi”. În mod silimar s-a tratat pentru numărul de rezervări.

Cererea 5:

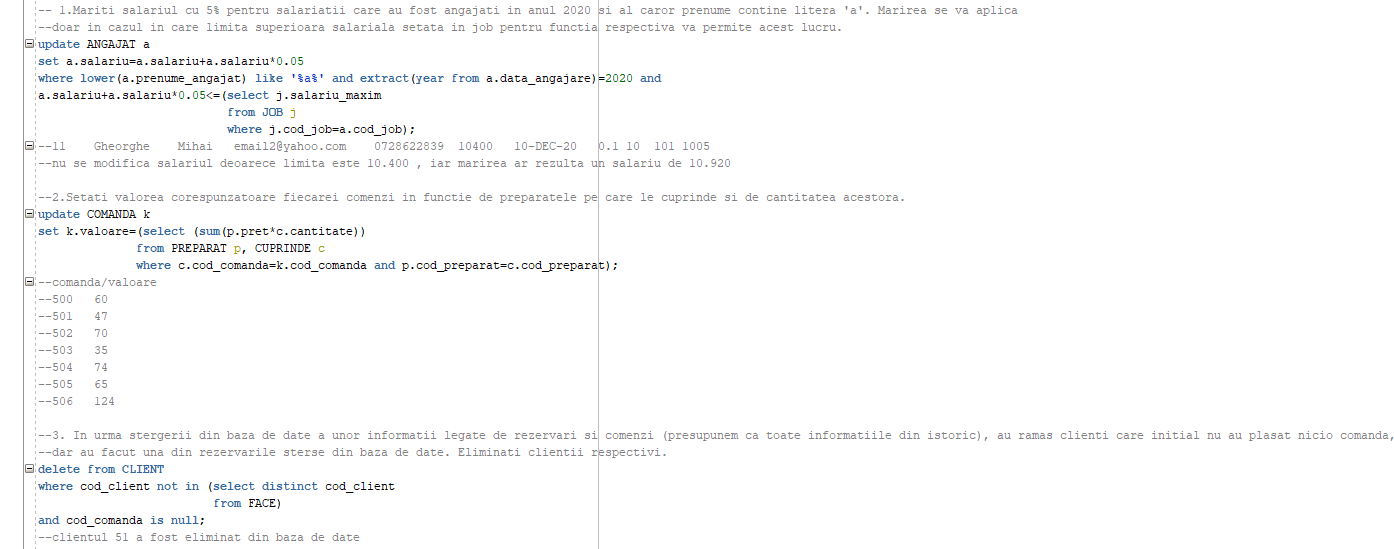


Pentru rezolvarea cererii, am utilizat o subcerere corelată care verifică dacă preparatul curent este cuprins într-o comandă. De asemenea, am verificat și celelalte condiții ale cerinței (preț>15 lei și numele clientului să conțină litera „a”).

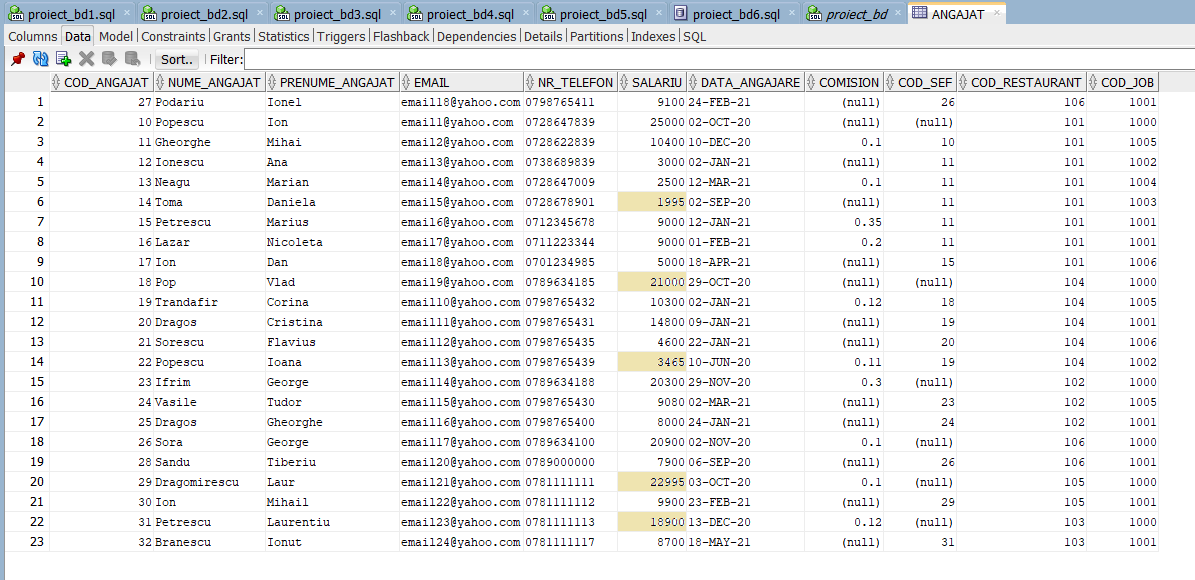
„Where exists” va verifica dacă există rezultat returnat de subcererea respectivă.

1. Implementarea a 3 operații de actualizare sau suprimare a datelor utilizând

subcereri.

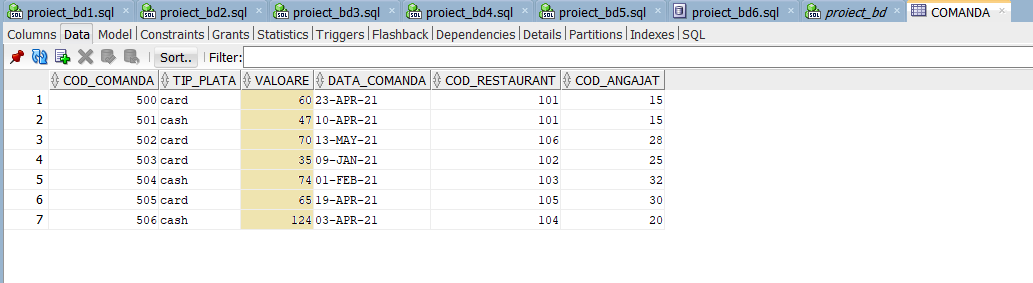


**ANGAJAT:**



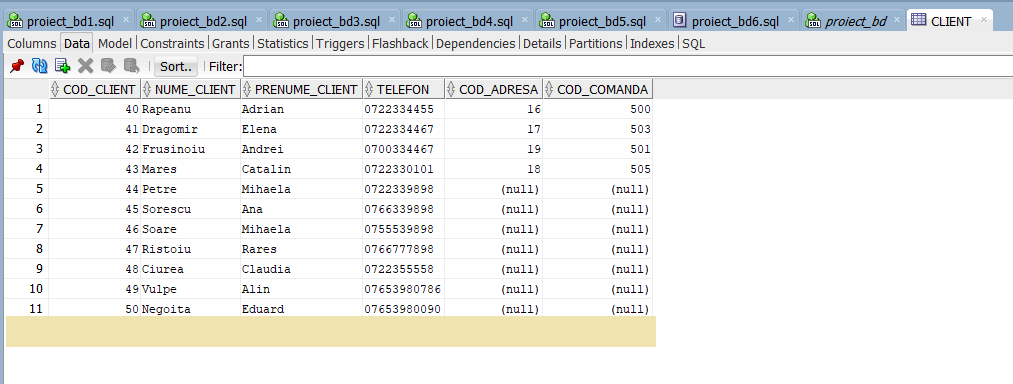
Salariații care au fost angajați în anul 2020 și au în componența prenumelui litera ”a” au primit o mărire de 5% în funcție de limita salarială a job-ului. Angajatul Gheorghe Mihai nu a beneficiat de aceasta mărire deoarece limita maximă salarială pentru postul său este de 10400, iar în urma măririi ar fi obținut 10920.

**COMANDA:**



Valoarea fiecărei comenzi a fost setată corespunzător, în funcție de prețul preparatelor pe care le cuprinde și de numărul acestora.

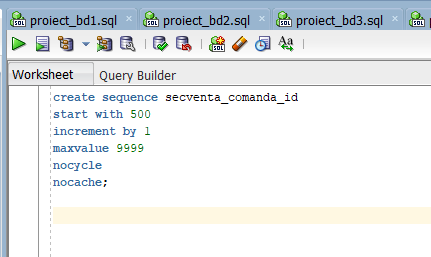
**CLIENT:**



Clientul 51 a fost eliminat deoarece inițial nu a făcut o comandă și a făcut o rezervare care nu mai există în baza de date.

1. Crearea unei secvențe ce va fi utilizată în inserarea înregistrărilor în tabele

(punctul 10).

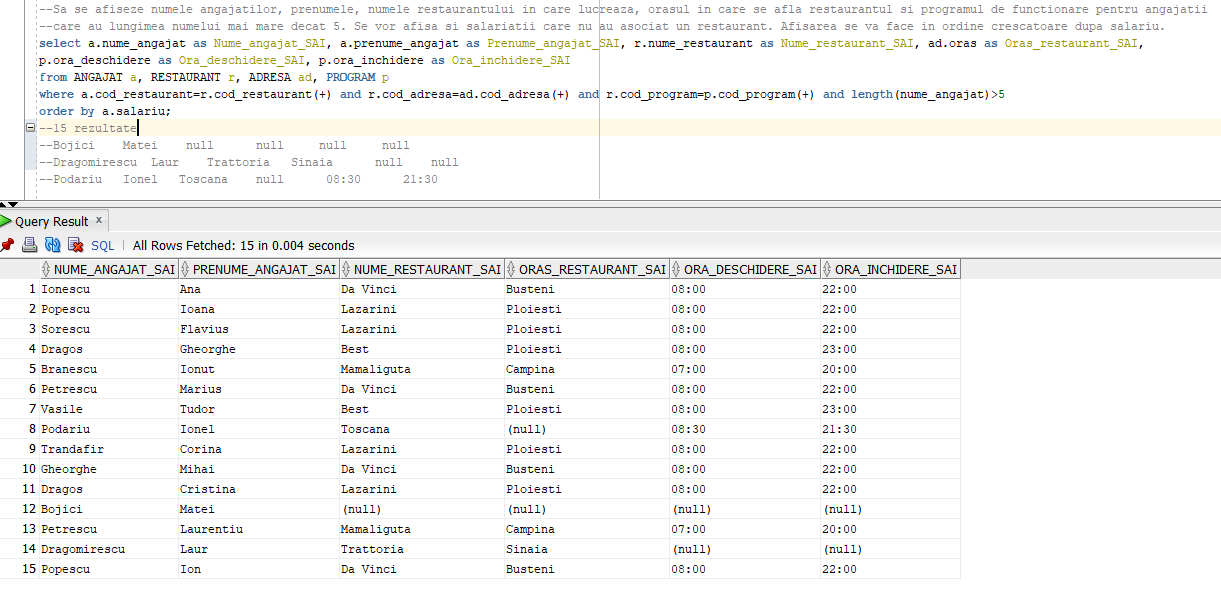


Secvența de mai sus a fost folosită la inserarea datelor în tabela COMANDA, mai exact în generarea codului comenzii. Acest cod a pornit de la valoarea 500, a fost incrementat cu 1, maximul fiind 9999 deoarece cod\_comanda este de tipul number(4).

1. Formulați în limbaj natural și implementați în SQL: o cerere ce utlizează

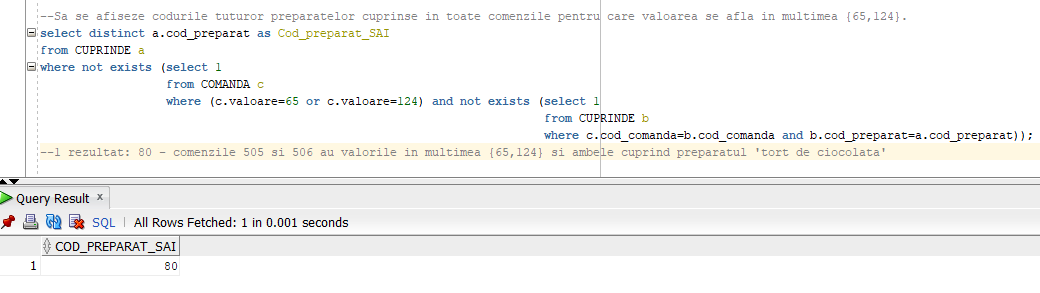
operația *outer-join* pe minimum 4 tabele și două cereri ce utilizează operația *division*.

Outer-join:

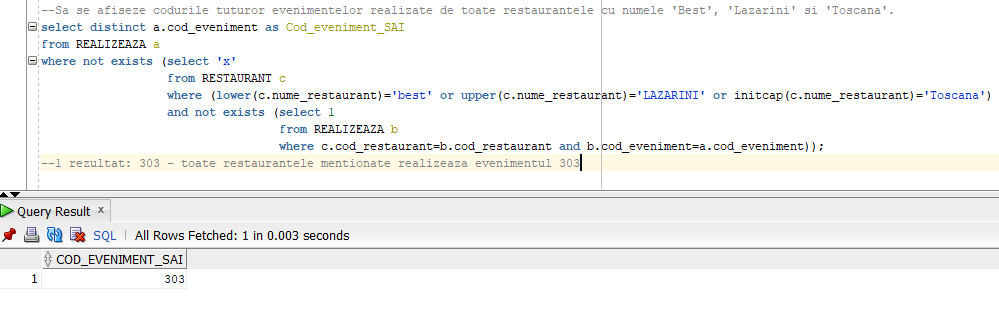


Pentru a afișa și angajații care nu au asociat un restaurant este necesar (+)/angajat a left outer join restaurant r/restaurant r right outer join angajat a (similar pentru adresa și program). În lipsa acestuia, salariații Bojici, Dragomirescu și Podariu nu s-ar afișa pentru că nu au restaurantul setat/restaurantul nu are adresa setată/restaurantul nu are programul setat. Plasăm (+) în dreptul coloanelor deficitare în informații. Se va poziționa și în dreptul codului de adresă și de program deoarece lipsa informațiilor din RESTAURANT se transmite mai departe în tabelele corespunzătoare.

Division:



În această cerere, am simulat operatorul division utilizând de două ori NOT EXISTS. Se vor afișa codurile preparatelor pentru care nu există comenzi cu valoarea indicată care să nu cuprindă preparatul respectiv. Am folsit select *constantă* deoarece nu este necesar ca instrucțiunea sa returneze o anumită valoare și este mai eficient din punct de vedere al performanței.



În cea de-a doua cerere am procedat în mod similar, utilizând de două ori NOT EXISTS și select *constantă*. Se vor afișa codurile evenimentelor pentru care nu există restaurant cu numele menționt în cerință care să nu realizeze evenimentul respectiv.

1. Optimizarea unei cereri, aplicând regulile de optimizare ce derivă din

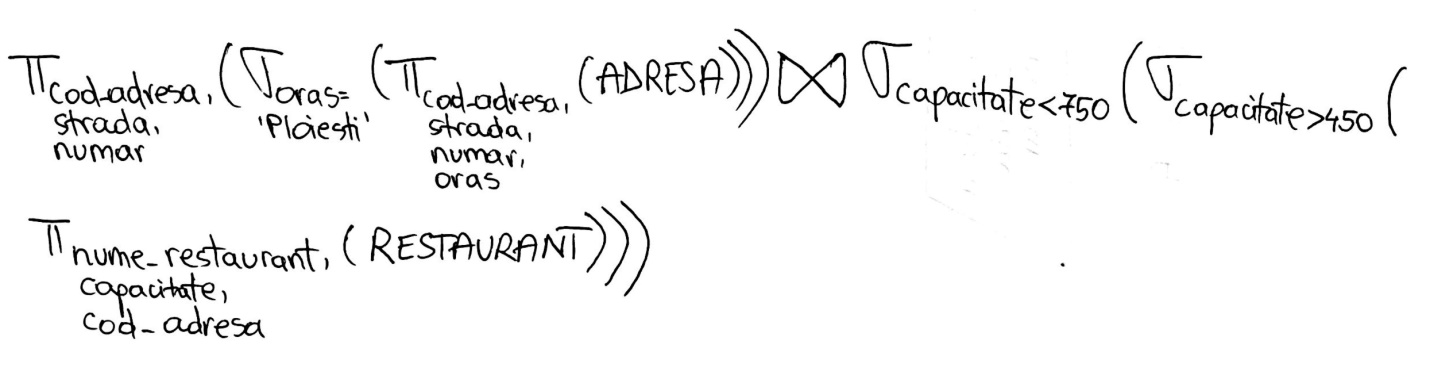
proprietățile operatorilor algebrei relaționale. Cererea va fi exprimată prin expresie algebrică, arbore algebric și limbaj SQL, atât anterior cât și ulterior optimizării.

*Cerință*: Să se afișeze codurile adreselor, strada, numărul, numele locației și capacitatea restaurantelor aflate în orașul Ploiești, cu capacitatea strict mai mare decât 450 și strict mai mică decât 750.

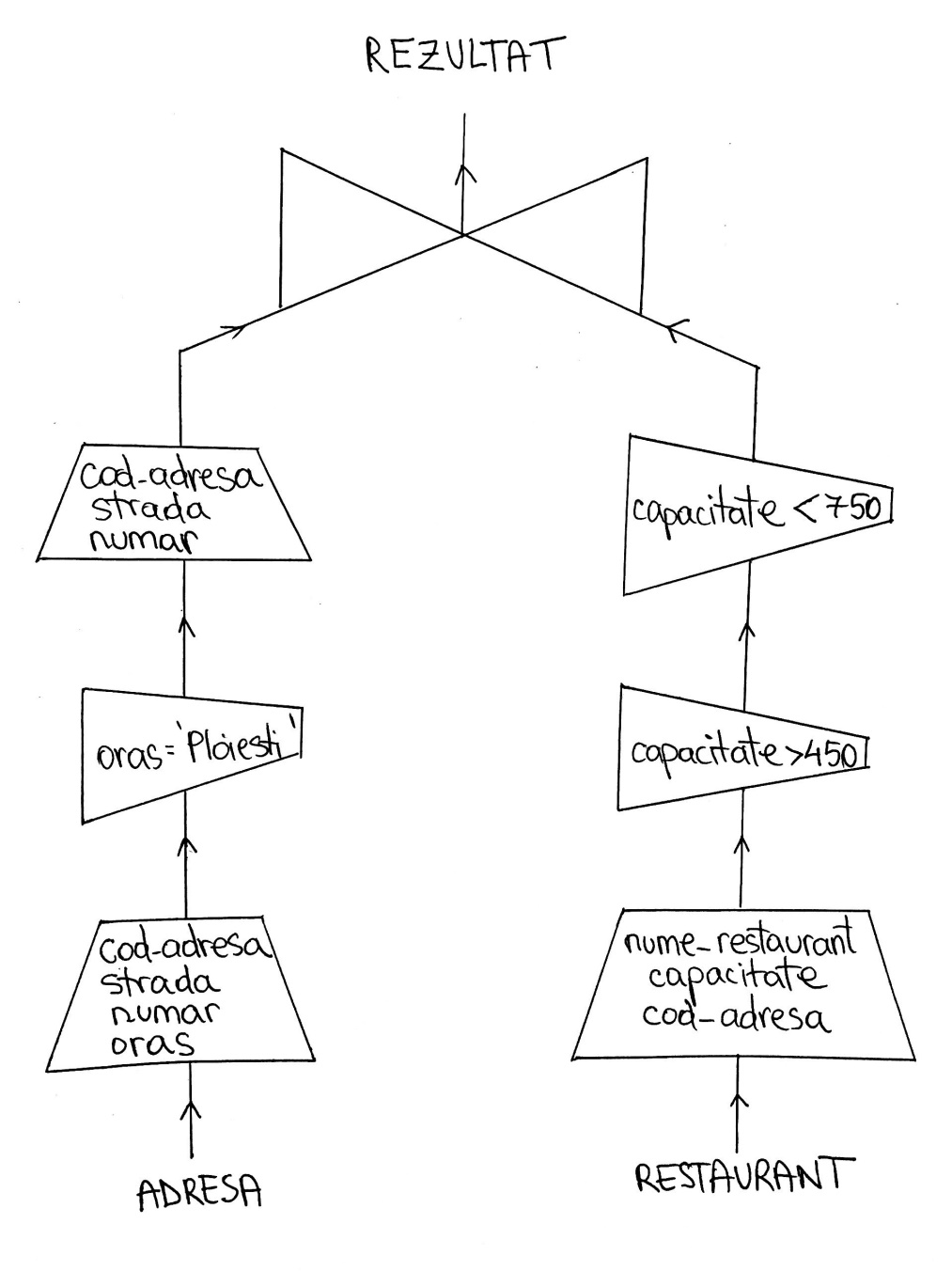
* R1 = PROJECT(ADRESA, cod\_adresa, strada, numar, oras);
* R2 = SELECT(R1, oras = ‘Ploiesti’);
* R3 = PROJECT(R2, cod\_adresa, strada, numar);
* R4 = PROJECT(RESTAURANT, nume\_restaurant, capacitate, cod\_adresa);
* R5 = SELECT(R4, capacitate>450);
* R6 = SELECT(R5, capacitate<750);
* REZULTAT = JOIN(R3, R6).

Rezultat inițial:

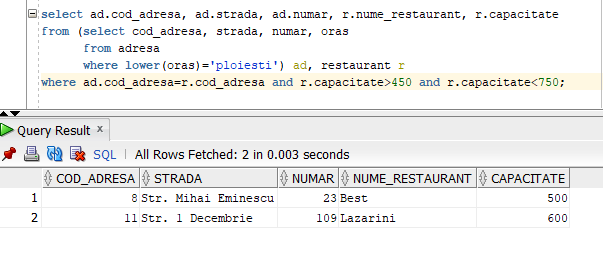
* Expresie algebrică:



* Arbore algebric:



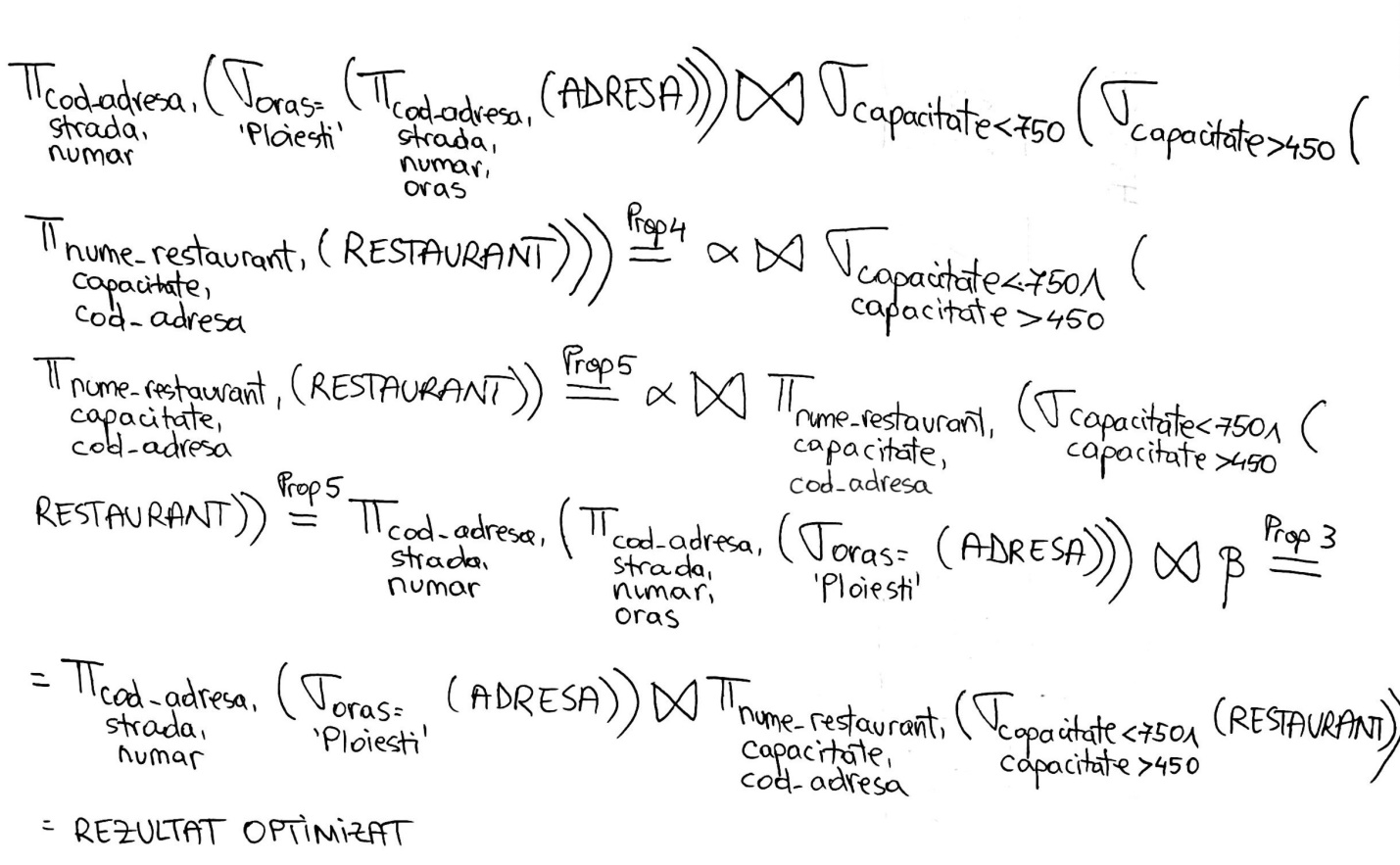
* SQL:



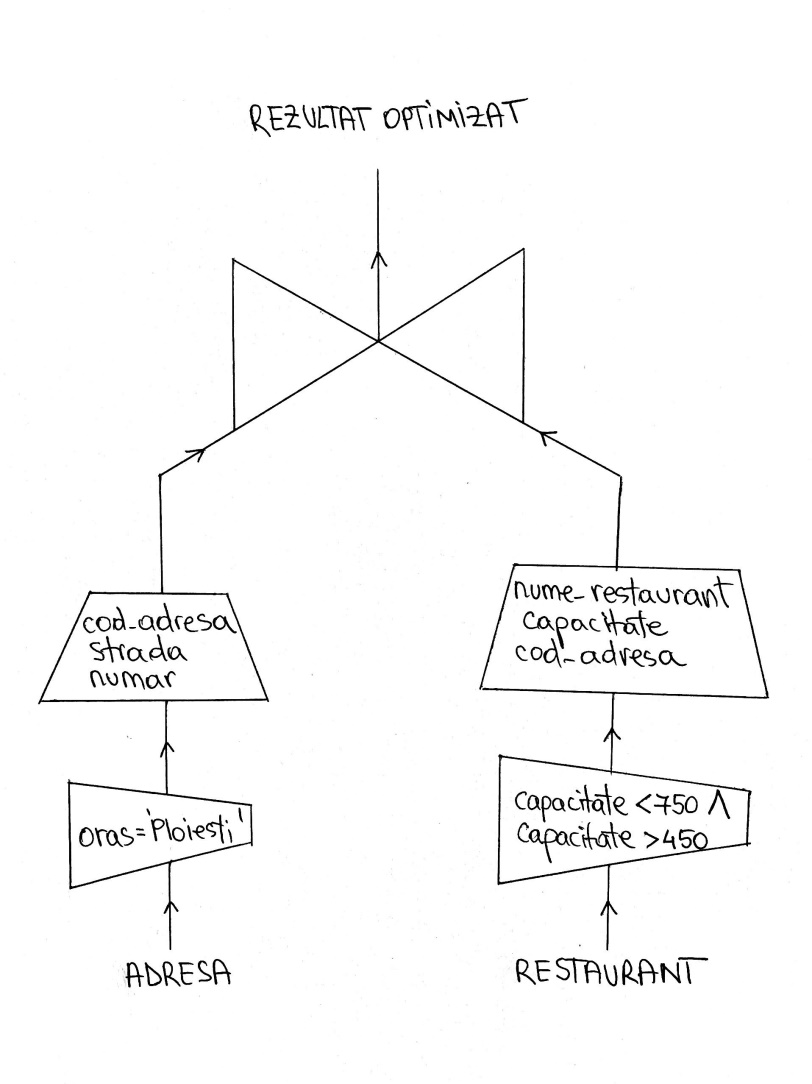
Număr de rezultate: 2.

Rezultat optimizat:

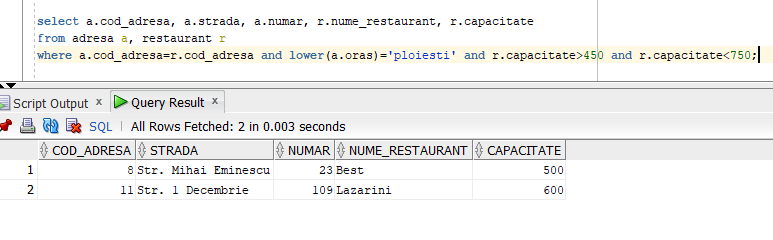
* Expresie algebrică:



* Arbore algebric:



* SQL:



Număr de rezultate: 2.

1. a) Realizarea normalizării BCNF, FN4, FN5. (**observație**: exemplele de

mai jos sunt relative la modelul proiectat)

* **BCNF**

O relație este în BCNF dacă și numai dacă fiecare determinant este o cheie candidat.

Vom presupune că un eveniment este de un anumit tip, dar facilitățile sunt unice în funcție de acest tip.

*Non BCNF:*

|  |  |  |
| --- | --- | --- |
| Cod\_eveniment# | Facilitate# | Tip\_eveniment |
| E1 | F1 | T1 |
| E2 | F1 | T1 |
| E3 | F2 | T2 |
| E4 | F3 | T3 |

Relația se va descompune în:

|  |  |
| --- | --- |
| Cod\_eveniment# | Tip\_eveniment |
| E1 | T1 |
| E2 | T1 |
| E3 | T2 |
| E4 | T3 |

|  |  |
| --- | --- |
| Tip\_eveniment# | Facilitate |
| T1 | F1 |
| T2 | F2 |
| T3 | F3 |

Relațiile sunt acum în BCNF, însă s-a pierdut dependența:

*cod\_eveniment, facilitate tip\_eveniment*

* **FN4**

O relație este în FN4 dacă și numai dacă este în BCNF și orice dependență multivaloare este o dependență funcțională.

*Non FN4:*

|  |  |  |
| --- | --- | --- |
| Cod\_restaurant# | Cod\_produs# | Cod\_zona\_distribuție# |
| R1 | P1 | Z1 |
| R1 | P1 | Z2 |
| R1 | P2 | Z1 |
| R1 | P2 | Z2 |
| R2 | P1 | Z3 |
| R2 | P2 | Z3 |
| R3 | P1 | Z1 |
| R3 | P1 | Z4 |
| R3 | P1 | Z5 |
| R3 | P2 | Z1 |
| R3 | P2 | Z4 |
| R3 | P2 | Z5 |

Din cauza faptului că produsele disponibile livrării sunt independente de zonele de distribuție, există redundanțe în tabel. Dacă dorim să mai adăugăm un produs nou pentru un anumit restaurant, va trebui să inserăm acest lucru de mai multe ori, pentru fiecare zonă de distribuție. Din acest motiv, tabelul se va descompune în două tabele, unul în care reținem restaurantul și produsul disponibil livrării și cel de-al doilea în care reținem restaurantul și zona de distribuție.

|  |  |
| --- | --- |
| Cod\_restaurant# | Cod\_produs# |
| R1 | P1 |
| R1 | P2 |
| R2 | P1 |
| R2 | P2 |
| R3 | P1 |
| R3 | P2 |

|  |  |
| --- | --- |
| Cod\_restaurant# | Cod\_zona\_distribuție# |
| R1 | Z1 |
| R1 | Z2 |
| R2 | Z3 |
| R3 | Z1 |
| R3 | Z4 |
| R3 | Z5 |

* **FN5**

O relație este în FN5 dacă și numai dacă aceasta este în FN4 și nu conține dependențe ciclice.

Vom presupune că un restaurant poate organiza mai multe evenimente. De asemenea, nu toate restaurantele organizează aceleași evenimente. Exemplu: E1 – eveniment de tip petrecere de weekend, E2 – petrecere privată.

*Non FN5:*

|  |  |  |
| --- | --- | --- |
| Cod\_restaurant# | Cod\_eveniment# | Data\_eveniment |
| R1 | E1 | 01-04-2022 |
| R2 | E1 | 01-04-2022 |
| R2 | E1 | 08-04-2022 |
| R2 | E2 | 08-04-2022 |

Relația este în FN4, dar există redundanțe în date.

R1(cod\_restaurant#, cod\_eveniment#) s-a eliminat (R2, E1)

R2(cod\_restaurant#, data\_eveniment) s-a eliminat (R2, 08-04-2022)

R3(cod\_eveniment#, data\_eveniment) s-a eliminat (E1, 01-04-2022)

|  |  |
| --- | --- |
| Cod\_restaurant# | Cod\_eveniment# |
| R1 | E1 |
| R2 | E1 |
| R2 | E2 |

|  |  |
| --- | --- |
| Cod\_restaurant# | Data\_eveniment |
| R1 | 01-04-2022 |
| R2 | 01-04-2022 |
| R2 | 08-04-2022 |

|  |  |
| --- | --- |
| Cod\_eveniment# | Data\_eveniment |
| E1 | 01-04-2022 |
| E1 | 08-04-2022 |
| E2 | 08-04-2022 |

b) Aplicarea denormalizării, justificând necesitatea acesteia.

Vom analiza relația dintre tabela PROGRAM și RESTAURANT, în cazul de denormalizare tabele de căutare.

PROGRAM(cod\_program#, descriere)

RESTAURANT(cod\_restaurant#, nume\_restaurant, capacitate, cod\_program)

Acest exemplu este în FN3.

În acest caz, avantajele sunt că se reduce dimensiunea relației de la capătul many al relației (RESTAURANT) și că dacă este modificată descrierea programului, atunci se va modifica o singura dată.

Necesitatea denormalizării apare în cazul în care putem considera că accesarea descrierii are loc adesea odată cu accesarea datelor despre restaurant. În acest caz, se va denormaliza în felul următor:

* Entitatea PROGRAM va dispărea complet;
* Entitatea RESTAURANT devine:

RESTAURANT(cod\_restaurant#, nume\_restaurant, capacitate, descriere\_program).

BIBLIOGRAFIE

Curs – Baze de Date - Lect. Dr. Marin Letiția Ana

<http://www.elth.pub.ro/~preda/teaching/BDE/BDE_5.pdf>

<http://vega.unitbv.ro/~cataron/Courses/BD/BD_Cap_4.pdf>