Raport Analiză

Metode de optimizare si distribuire in Baze de Date

Prof. Coordonator Lectr. dr.Gabriela Mihai

Grupa 506

Studenti:

Badea Bogdan-Andrei

Balica Adrian-Claudiu

Mancila Doru-Bogdan

Universitatea din Bucuresti

2023

Cuprins

1. Descrierea modelului

Aplicația este destinată unui lanț de restaurante cu scopul de a gestiona atât angajații cât și meniul și activitatea lor din restauratele respective. Mai concret, aplicația va gestiona locația restaurantelor, orașul din care face parte, tipurile de angajați, care este customizabil, angajați care fac parte dintr-un anumit restaurant, meniurile alături de băuturile și mâncărurile aferente acestuia și în final gestionarea dintre angajați și comenzi, angajatul putând fi chelner și a prelua o comandă sau bucătar și să aibă misiunea de a găti felul de mâncare corespunzător. De asemenea, fiecărui order ii vor fi calculate și recalculate automat valoarea unei comenzi prin adăugarea sau editarea numărului de băuturi sau feluri de mâncare de pe comandă.

2. Diagramele bazei de date OLTP

În urmă prezentării aplicației și descriere și scopul aplicației,structura bazei de date va conține următoarele tabele:

- ***Types***: In types se vor stoca tipurile de angajați posibil, astfel făcând dinamic structura tipurile de angajați
- ***Employees***: tabela în care se stochează efectiv angajații care lucrează în lanțul de restaurante
- ***Cities***: tabela ce stochează orașele în care avem restaurante din lanțul nostru
- ***Restaurants***: tabela ce reține restaurantele create din lanțul de restaurante
- ***Orders***: tabela ce stochează comenzile făcute în restaurante
- ***Menus***: tabela ce stochează menurile
- ***Drinks***: tabela ce conține toate băuturile disponibile din lanțul de resturante
- ***Dishes***: tabela ce conține toate felurile de mâncare disponibile din lanțul de resturante

Schema OLTP prezintă următoarele relații:

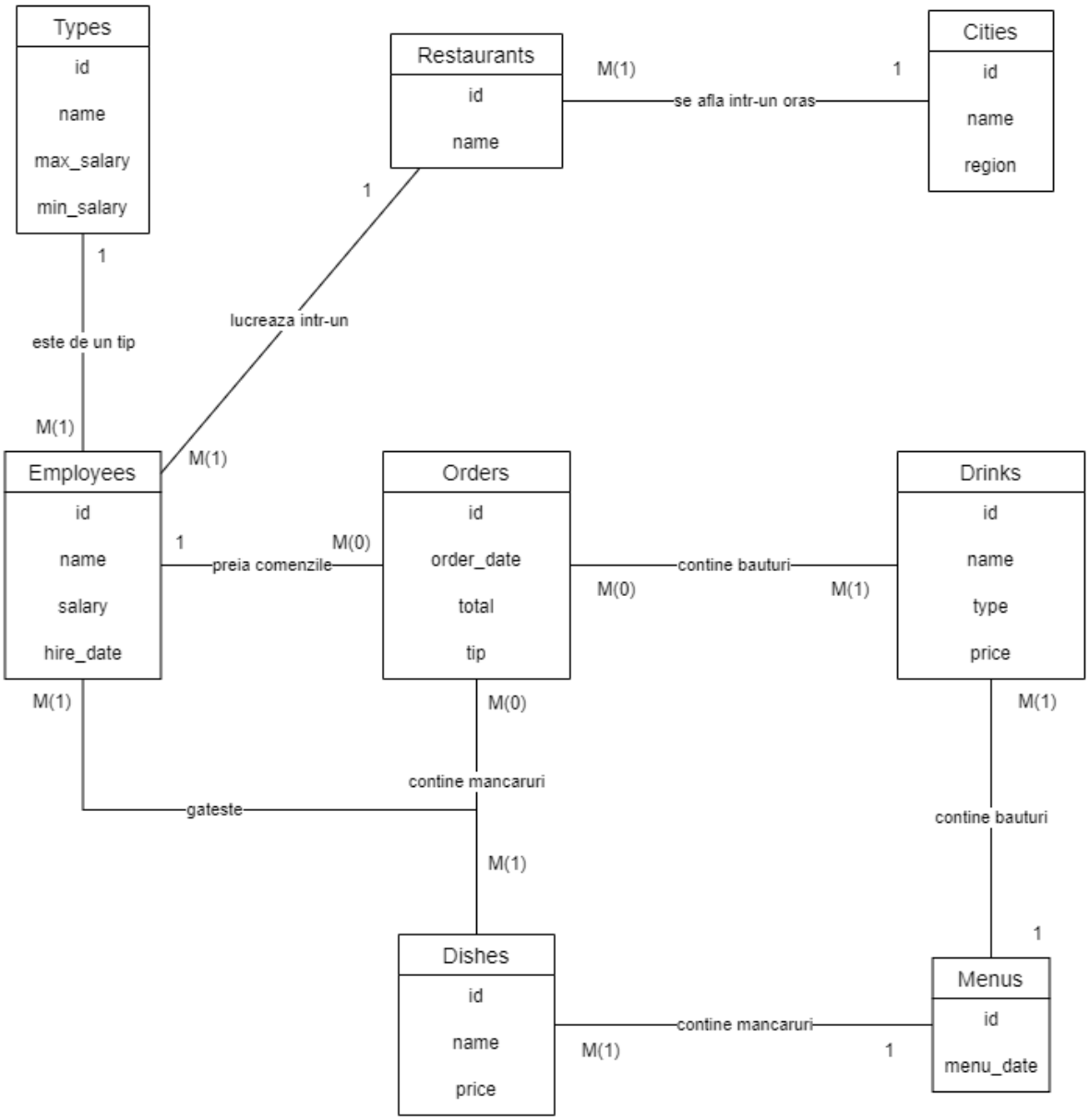
- ***Types*** – ***Employees***, relație de tip One-to-Many (1-M),deoarece mai mulți angajați pot avea același tip
- ***Restaurants*** – ***Employees***, relație de tip One-to-Many (1-M), restaurantele au mai mulți angajați
- ***City*** – ***Restaurants***, relație de tip One-to-Many (1-M), orasele au mai multe restaurante
- ***Employees*** – ***Employees - Dishes*** , relație de tip "3", ce va fi mapată cu ajutorul unei tabele associative care conține chei străine către toate tabele. Acest lucru a fost ales deoarece că avem tipuri de angajați posibili și practic în tabela asociativă vom avea trei relații One-To-Many de la tabela nou creată la cele de baza și această va conține chef_id, order_id, dish_id pentru a face mai ușoară obținerea mâncării din comenzi și cine a gătit-o și va conține și count și price pentru calcularea totalului
- ***Employees*** – ***Orders***, relație de tip One-to-Many (1-M), un employees va avea mai multe ordere, relație străînă in acest caz fiind waiter_id, persoană care preia comandă
- ***Orders*** – ***Drinks***, relație de tip Many-to-Many (M-M) – care va fi rezolvată cu o tabela asociativă orders_drinks care conține adițional înafara de chei campurie price (suma totală a comenzii pe băuturi price drink*count) și count (câte băuturi de un anumit tip sunt)
- ***Menus*** – ***Drink***, relație de tip One-to-Many (1-M), meniul conține mai multe bauturi

- ***Menus*** – ***Dishes***, relație de tip One-to-Many (1-M), meniul conține mai tipuri de mancare

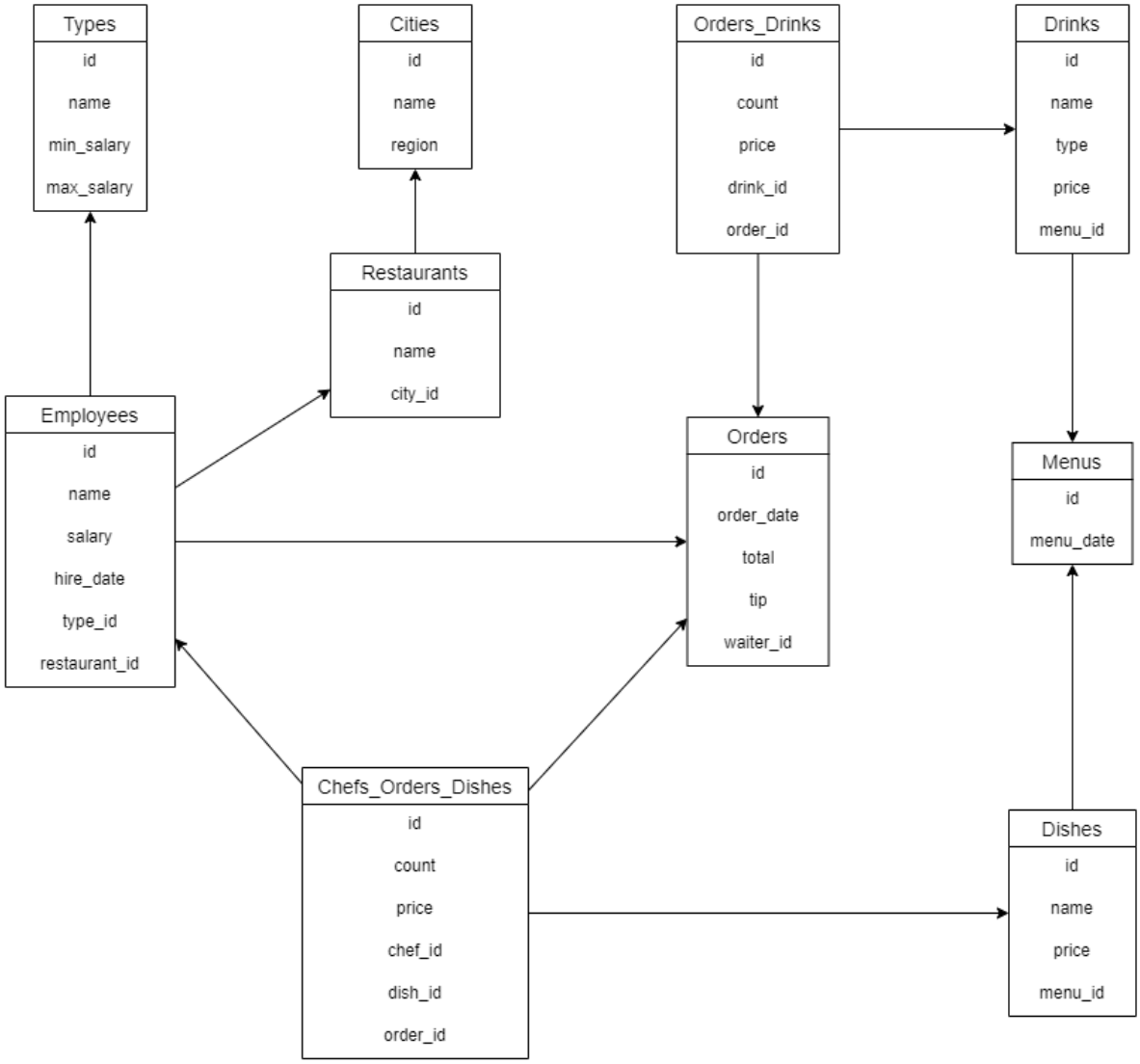
Au fost creați trei utilizatori pentru împărțirea permisiunilor:

- *User_mobd*: utilizatorul pentru cele două scheme obținute în urma fragmentarii orizontale
- *user_mobd_global*: utilizatorul responsabil de schema globală
- *user_modbd_centralizat*: utilizatorul schemei conceptuale

2.1 Diagrama enitate-relație



2.2 Schema conceptuală



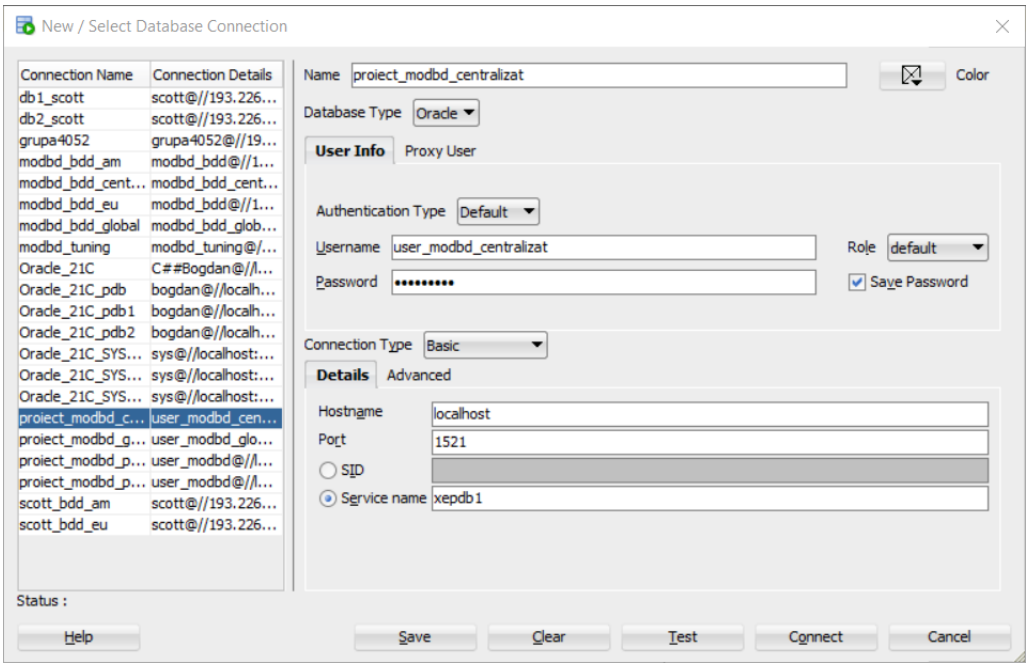
3. Descrierea modului distribuire

Baza de date centralizată este reprezentată de baza de date descrisă în capitolul anterior. Baza de date distribuită este gestionată prin intermediul aceluiași sistem de gestiune SGBD - sqldeveloper.

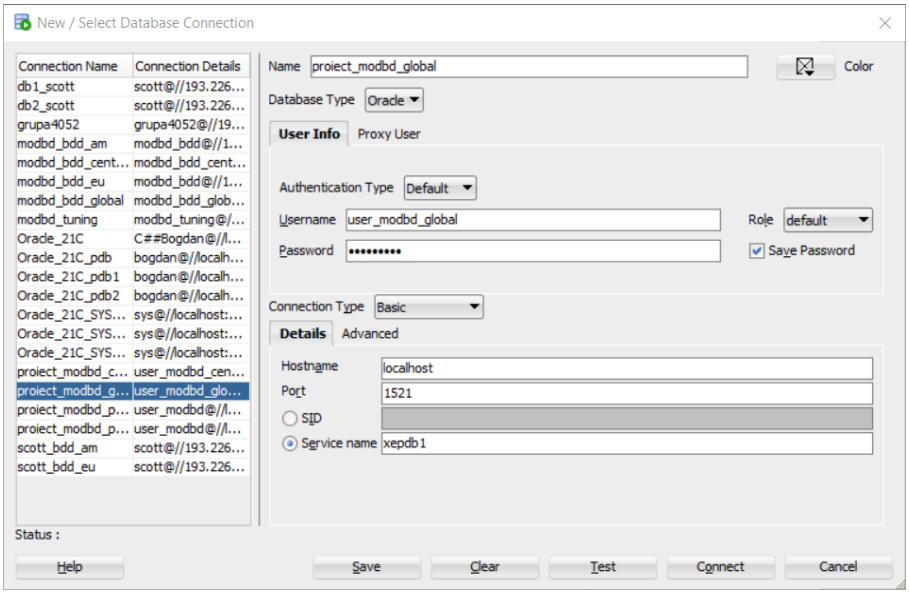
Bazele de date sunt createa local folosind oracle21C. Prin intermediul Sqldeveloper, am create două baze de date de tip pluggable, numite XEPDB1 și XEPDB2. Au fost create conexiuni diferite pentru a simula și crea practice logică pentru bazele de date distribuite. Acestea sunt împărțite în submultimi disjuncte pentru a face posibilă practic fragmentarile atât pe orizontală cât și pe verticală. Pentru a putea prelua toate datele și a crea view cu acestea se folosins link-uri precum “cities_prov@pdb2”.

Fiind la opțional fragmentare verticală s-a făcut doar ilustrativ pe tabela types iar cea verticală se face în funcția de regiunea orașului, această fiind făcute pe câmpul regiune după “capitală” și “provincie”.

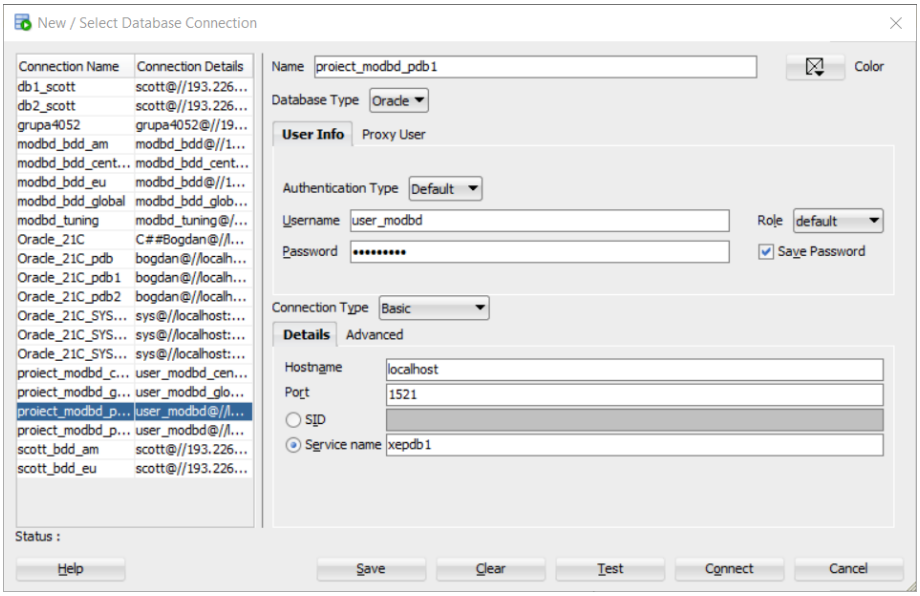
În pdb1 avem informațiile pentru ”capitală” iar în pdb2 cele pentru “provincie”. Fragmentarea verticală făcută pe tabela types a fost făcută în felul următor: în type_cap avem id și nume iar în type_prov avem id, min_salary și max_salary în funcție de frecvența utilizării lor



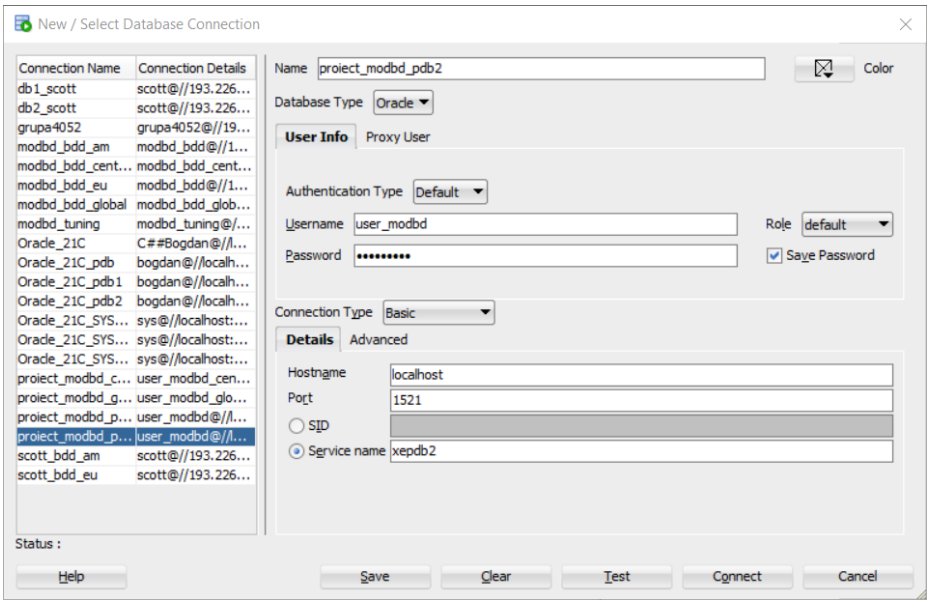
Conectare la baza de date centralizată



Conectare la baza de date globală



Conectare la pdb1



Conectare la pdb2

4. Fragmentarea relațiilor

Fragmentarea relațiilor în bazele de date se referă la împărțirea datelor unei tabele în mai multe fragmente, astfel încât fiecare fragment să fie stocat pe un server sau o locație diferită. Acest proces poate fi util în situațiile în care o bază de date este prea mare pentru a fi stocată pe un singur server sau ne dorim performante mult mai bune pentru o baza de date foarte vasta. Beneficiile acesteia ar fi securitatea datelor, eficienta, se pot optimiza cereri sql mai usor, insa avem si cateva puncta mai putin bune pentru fragmentare precum viteza de accesare daca avem prea multe fragmente sau poate fi foarte cositistoare daca folosim o fractmentare recursiva.

4.1 Fragmentare orizontala

Fragmentarea orizontală face referire la procesul de împărțire a datelor, având aceleași coloane dar rânduri diferite în funcție de factorul după care se face fragmentarea, în cazul nostru ar fi câmpul region din tabela city, iar fragmentarea s-ar face între valorile acesteia, capitală și provincie.

Pentru pdb1 avem fragmentarea orizontală realizată pentru înregistrările ale căror valoare pentru câmpul ‘region’ este ‘capital’, iar pentru pdb2 pentru valoarea ‘province’ a regiunii

- PDB1: region = ‘capital’
- PDB2: region = ‘province’

4.2 Fragmentare orizontala derivate

Fragmentarea orizontală derivată se referă la împărțirea tabelei în fragmente, dar aceste fragmente nu conțîn aceleași rânduri că și tabela originală. În schimb, aceste fragmente sunt create prin aplicarea unor criterii sau reguli de selecție a unui subset specific de rânduri din tabela original, în cazul nostrum se va aplică în funcție de regiune.

1. Fie L legătura dintre relațiile Cities și Restaurants

$$owner(L) = CITIES$$
$$member(L) = RESTAURANTS$$

În funcție de cities putem grupa restaurantele în 2 fragmente: resturante aflate in orase din regiunea 'province' sau restaurante aflate in roas din regiunea 'capital'

$$RESTAURANTS_1 = RESTAURANTS \ltimes CITIES_1$$
$$RESTAURANTS_2 = RESTAURANTS \ltimes CITIES_2$$

unde

$$CITIES_1 = \sigma_{cites-region\,=\,capital}(CITIES)$$
$$CITIES_2 = \sigma_{cites-region\,=\,province}(CITIES)$$

2. Fie L legătura dintre relațiile Restaurants și Employees

$$owner(L) = RESTAURANTS$$
$$member(L) = EMPLOYEES$$

În funcție de restaurants putem grupa restaurantele în 2 fragmente: employees care lucreaza in restaurante din orase din regiunea 'province' sau din regiunea 'capital'

$$EMPLOYEES_1 = EMPLOYEES \ltimes RESTAURANTS_1$$
$$EMPLOYEES_2 = EMPLOYEES \ltimes RESTAURANTS_2$$

unde

$$RESTAURANTS_1 = \sigma_{cites-region\,=\,capital}(RESTAURANTS)$$
$$RESTAURANTS_2 = \sigma_{cites-region\,=\,province}(RESTAURANTS)$$

3. Fie L legătura dintre relațiile Employees și Orders

$$owner(L) = EMPLOYEES$$
$$member(L) = ORDERS$$

În funcție de employyes putem grupa orderele în 2 fragmente: ordere prelucrate de angajati din orase din regiunea 'province' sau din regiunea 'capital'

$$ORDERS_1 = ORDERS \ltimes EMPLOYEES_1$$
$$ORDERS_2 = ORDERS \ltimes EMPLOYEES_2$$

unde

$$EMPLOYEES_1 = \sigma_{cites-region\,=\,capital}(EMPLOYEES)$$
$$EMPLOYEES_2 = \sigma_{cites-region\,=\,province}(EMPLOYEES)$$

4. Fie L legătura dintre relațiile Orders și Orders_dishes

$$owner(L) = ORDERS$$

$$member(L) = ORDERS\_DISHES$$

În funcție de order putem grupa order_dishes în 2 fragmente: ordere_dishes ce fac parte din ordere ale caror regiune fac parte din 'province' sau din 'capital'

$$ORDERS\_DISHES_1 = ORDERS\_DISHES \ltimes ORDERS_1$$
$$ORDERS\_DISHES_2 = ORDERS\_DISHES \ltimes ORDERS_2$$

unde

$$ORDERS_1 = \sigma_{cites-region\,=\,capital}(ORDERS)$$
$$ORDERS_2 = \sigma_{cites-region\,=\,province}(ORDERS)$$

5. Fie L legătura dintre relațiile Orders și Chefs_orders_dishes

$$owner(L) = ORDERS$$
$$member(L) = CHEFS\_ORDERS\_DISHES$$

În funcție de orders putem grupa chefs_orders_dishes în 2 fragmente: chefs_orders_dishes care fac parte din ordere realizate in 'province' sau 'capital'

$$CHEFS\_ORDERS\_DISHES_1 = CHEFS\_ORDERS\_DISHES \ltimes ORDERS_1$$
$$CHEFS\_ORDERS\_DISHES_2 = CHEFS\_ORDERS\_DISHES \ltimes ORDERS_2$$

unde

$$ORDERS_1 = \sigma_{cites-region\,=\,capital}(ORDERS)$$
$$ORDERS_2 = \sigma_{cites-region\,=\,province}(ORDERS)$$

4.3 Fragmentare verticala

Fragmentarea verticală în bazele de date se referă la împărțirea coloanelor unei tabele în fragmente, astfel încât fiecare fragment să conțină un subset specific de coloane. Acest proces poate fi util în situațiile în care tabela conține mai multe coloane și doar un subset mic de coloane este necesar pentru a fi accesat frecvent. Acest lucru este realizat pentru a reduce timpul de interogare pentru un volum mare de date în cazul în care avem tabele cu multe coloane des interogate.

În cazul nostru fiind la opțional, această fragmentare a fost realizată pur demonstrative fragmentat astfel tabelul types în 2: unul să conțină id și numele tipului de angajat, iar celălalt să conțină id-ul, min_salary și max_salary. Acest lucru se realizează pentru reducerea memoriei care trebuie stocate pentru baza de date, astfel această putând fi împărțită pe mai multe servere.

5.Verificarea fragmentarii

Verificarea corectitudinii este realizată atât pentru fragmentarea orizontală cât și pentru cea verticală, în acest sens fiind analizate cele trei reguli de corectitudine: completitudinea, reconstrucția și disjuncția.

Completitudinea presupune fragmentarea relației inițiale fără a suferi pierderi de informație.

Reconstrucția unei relații din fragmentele sale este realizată cu ajutorul operatorului de reuniune.Relația R (reconstructia) și mulțimea fragmentelor sale $F_R = \{R_1, R_2\}$ rezultă că:

$$R = \bigcup_{i=1}^{2} R_i \,, \forall\, R_i \in F_r$$

Disjuncția este asigurată dacă predicatele compuse care determină fragmentarea se exclud reciproc, cu alte cuvinte, dacă un articol ce se regăsește într-un fragment nu se regăsește și în altul.

5.1 Fragmentare orizontala

Fragmentarile au la baza urmatoarele:

$$P_r = \{p_1, p_2\}$$

p1: region = ‘capital’

p2: region = ‘province’

Cities

Completitudinea – CAP

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

Completitudinea – PROV

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

Reconstructia – CAP

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

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

Reconstructia – PROV

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

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

Disjuncția – CAP

```
SELECT * FROM cities_cap
INTERSECT
SELECT * FROM cities_prov@pdb2;
```

Disjuncția – PROV

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

Pentru toate celelalte fragmentări avem aceeași logică și cod de mai sus doar că modificat pentru fiecare entitate în parte: employees, restaurants, orders, orders_drinks, chefs_orders_dishes. În schimb pentru cea verticală, types avem următoarea verificare

5.2 Fragmentare verticala

Această este făcută pur demonstrativ fiind la opțional și a fost realizată pe tabela types:

Types

Completitudinea – CAP

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

Completitudinea – PROV

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

Reconstructia – CAP

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

Reconstructia – PROV

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

Disjuncția – CAP

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

Disjuncția – PROV

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

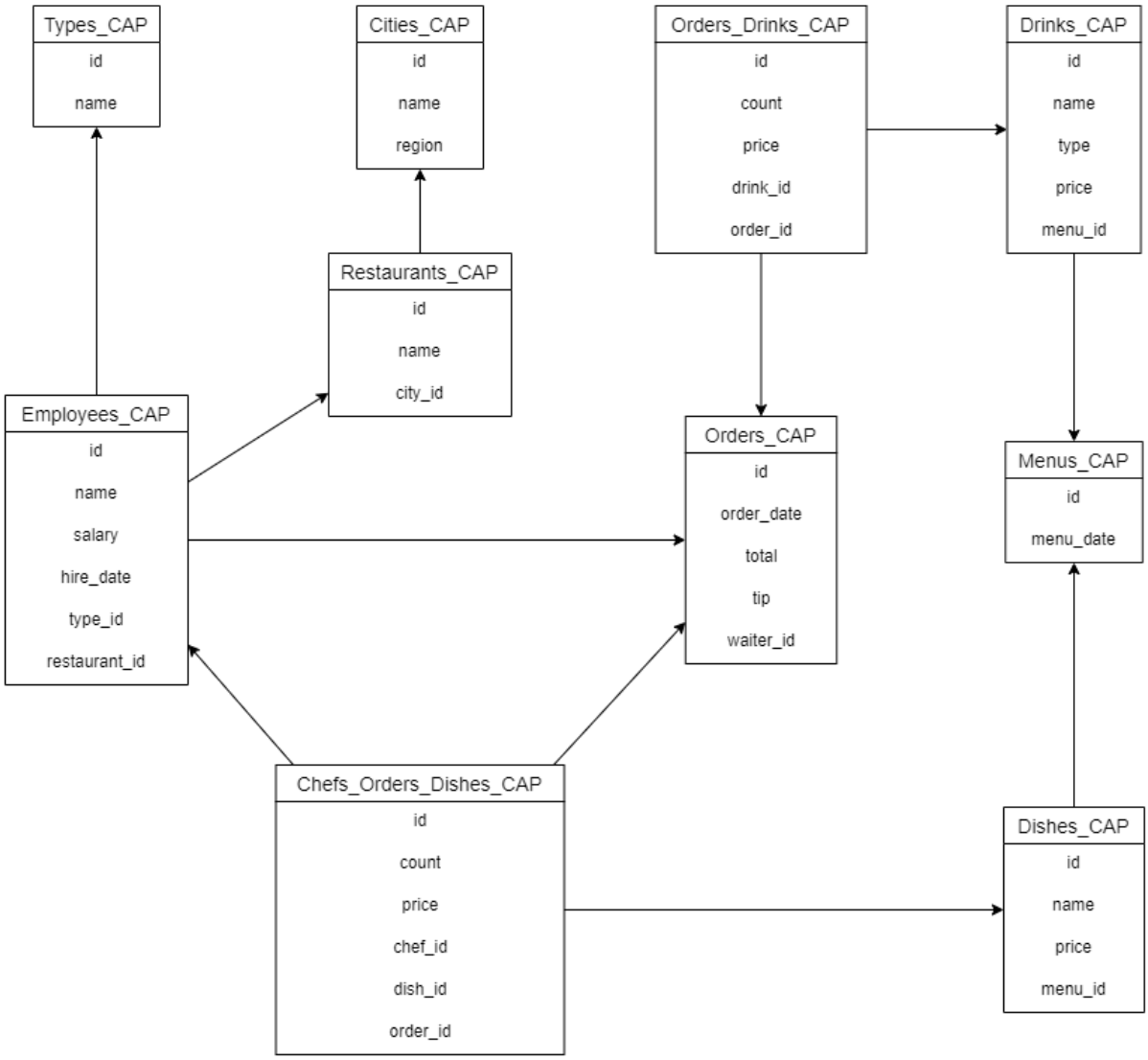
6.Decizia de replicare

Legat de replicare, această este utilă pentru tabelele de menu, drink și dishes deoarece fiind un lanț de restaurante acestea sunt date identice pentru toate restaurantele. Astfel este nevoie să se păstreze datele actualizate în orice moment și să avem aceeași informații indiferent de ce baza de date folosim.

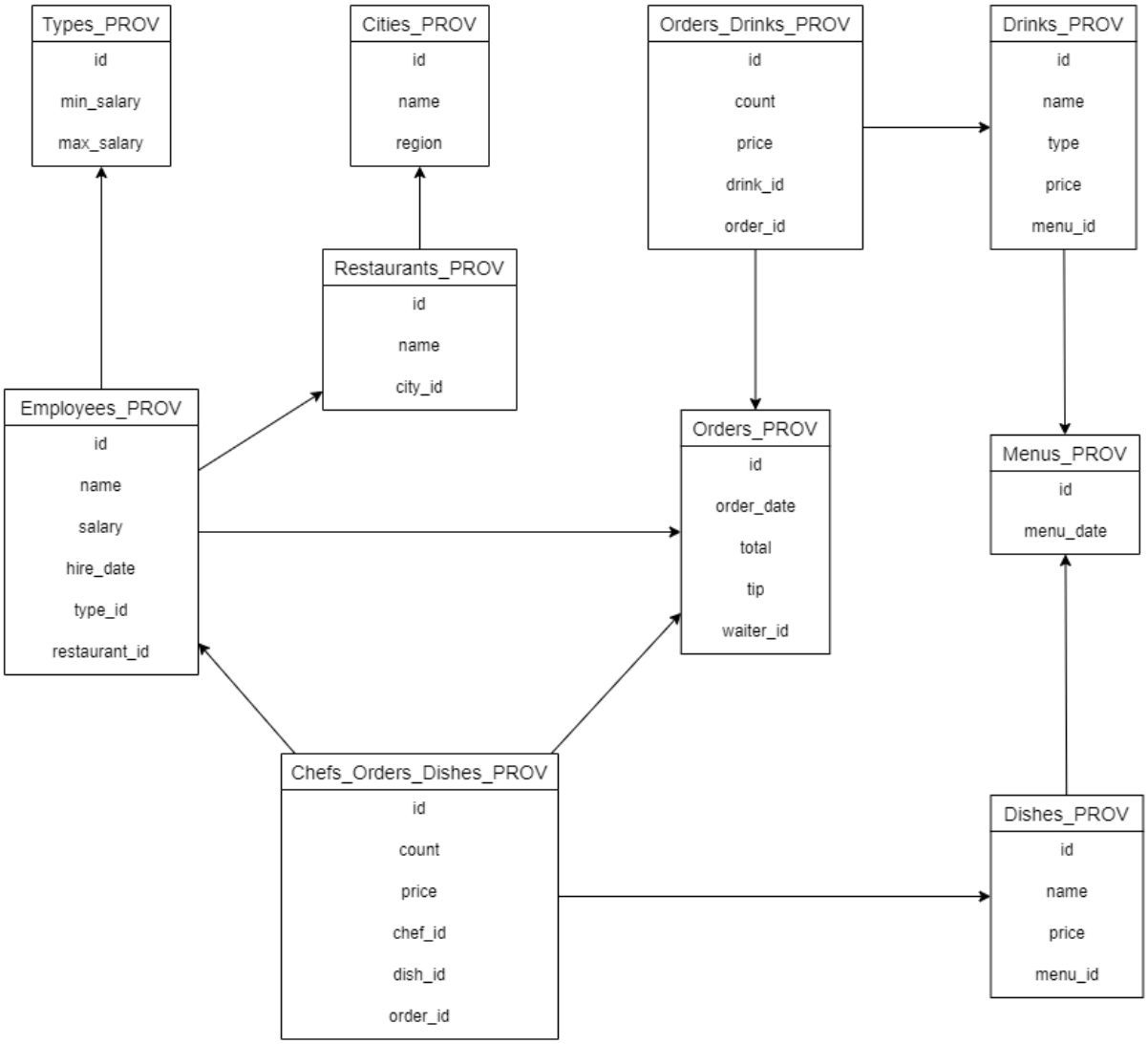
Acest lucru este realizat prin utilizarea triggerelor ce mențin datele actualizate în ambele baze de date atât când se fac actualizări cât și noi inserturi sau se șterg date.

7.Schemele conceptuale locale

Cele două scheme locale conceptuale sunt intitulate CAP și POV. Acestea provin datorită fragmentării orizontale care se realizează pe baza capului “region” din tabela city. Acesta poate fi provincie sau capitală. Cât pentru partea de de fragmentare verticală, fiind la opțional am decis să fie făcut doar islustrativ pe tabela Types și să împărțim în două elemente (id,name și id,min_salary, max_salary).



Schema conceptuala pentru capitala



Schema conceptuala provincie

8.Constrangeri

Bazele de date distribuite sunt eficiente pentru un flux mare de date, iar separarea lor în diferite clustere pentru eficientizare este una dintre cele mai bune soluții pentru a îmbunătății performațele. Însă o dată cu acest lucru apar și alte aspecte care trebuie luate în considerare precum integritatea datelor. Un bun exemplu ar fi cheile străine, dacă legăturile sunt făcute necorespunzător și sunt prea târziu detectate aceste probleme este foarte greu să remediezi astfel de probele fără un back-up existent. De asemenea, aceeași problema poate există și cu cheile primare care să nu corespundă. Însă acest back-up în cazul bazelor de date distribuite este destul de costisitor deoarece ai mai multe baze de date și nu doar una replicată. Astfel constrângerile aplicate de noi sunt la nivel de cheie primară și cheie străină.

Constrangeri de cheie primară:

- Pentru CAP:
  - Contrangerea campului id ca si primary key, pk_cities_cap in tabelul cities_cap
  - Contrangerea campului id ca si primary key, pk_restaurants_cap in tabelul resturants_cap
  - Contrangerea campului id ca si primary key, pk_types_cap in tabelul types_cap
  - Contrangerea campului id ca si primary key, pk_employees_cap in tabelul employees_cap
  - Contrangerea campului id ca si primary key, pk_menus_cap in tabelul menus_cap
  - Contrangerea campului id ca si primary key, pk_drinks_cap in tabelul drinks_cap
  - Contrangerea campului id ca si primary key, pk_dishes_cap in tabelul dishes_cap
  - Contrangerea campului id ca si primary key, pk_orders_cap in tabelul orders_cap
  - Contrangerea campului id ca si primary key, pk_chefs_orders_dishes_cap in tabelul chefs_orders_dishes_cap
  - Contrangerea campului id ca si primary key, pk_orders_cap in tabelul orders_cap
- Pentru PROV
  - Contrangerea campului id ca si primary key, pk_cities_prov in tabelul cities_prov
  - Contrangerea campului id ca si primary key, pk_restaurants_prov in tabelul resturants_prov
  - Contrangerea campului id ca si primary key, pk_types_prov in tabelul types_prov
  - Contrangerea campului id ca si primary key, pk_employees_prov in tabelul employees_prov
  - Contrangerea campului id ca si primary key, pk_menus_prov in tabelul menus_prov

- Contrangerea campului id ca si primary key, pk_drinks_prov in tabelul drinks_prov
- Contrangerea campului id ca si primary key, pk_dishes_prov in tabelul dishes_prov
- Contrangerea campului id ca si primary key, pk_orders_prov in tabelul orders_prov
- Contrangerea campului id ca si primary key, pk_chefs_orders_dishes_prov in tabelul chefs_orders_dishes_prov
- Contrangerea campului id ca si primary key, pk_orders_prov in tabelul orders_prov

Pe fiecare primary key setat avem constrângere și de unicitate globală creată printr-un trigger de genul acesta:

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

- Constrangeri de cheie straină:
  - Pentru CAP
    - Contrangerea campului city_id cu campul primary key ‘id’ de la cities_cap pentru restaurants_cap- fk_restaurants_cap_cities_cap
    - Contrangerea campului restaurant_id cu campul primary key ‘id’ de la restaurants_cap pentru employees_cap- fk_employees_cap_restaurants_cap
    - Contrangerea campului type_id cu campul primary key ‘id’ de la types_cap pentru employees- fk_employees_cap_type_cap
    - Contrangerea campului menu_id cu campul primary key ‘id’ de la menus_cap pentru drinks_cap - fk_drinks_cap_menus_cap
    - Contrangerea campului menu_id cu campul primary key ‘id’ de la menus_cap pentru dishes_cap -fk_dishes_cap_menus_cap
    - Contrangerea campului waiter_id cu campul primary key ‘id’ de la employees pentru orders_cap- fk_orders_cap_employees_cap
    - Contrangerea campului order_id cu campul primary key ‘id’ de la orders_cap pentru orders_drink_cap -fk_orders_drinks_cap_orders_cap
    - Contrangerea campului drink_id cu campul primary key ‘id’ de la drinks_cap pentru orders_drinks -fk_orders_drinks_cap_drinks_cap

- Contrangerea campului order_id cu campul primary key ‘id’ de la orders_cap pentru chefs_orders_dishes_cap - fk_chefs_orders_dishes_cap_orders_cap
- Contrangerea campului dish_id cu campul primary key ‘id’ de la dishes_cap pentru chefs_orders_dishes_cap - fk_chefs_orders_dishes_cap_dishes_cap
- Contrangerea campului chef_id cu campul primary key ‘id’ de la employee pentru chefs_orders_dishes_cap - fk_chefs_orders_dishes_cap_employee_cap

- Pentru POV
  - Contrangerea campului city_id cu campul primary key ‘id’ de la cities_prov pentru restaurants_prov- fk_restaurants_prov_cities_prov
  - Contrangerea campului restaurant_id cu campul primary key ‘id’ de la restaurants_prov pentru employees_prov- fk_employees_prov_restaurants_prov
  - Contrangerea campului type_id cu campul primary key ‘id’ de la types_prov pentru employees- fk_employees_prov_type_prov
  - Contrangerea campului menu_id cu campul primary key ‘id’ de la menus_prov pentru drinks_prov - fk_drinks_prov_menus_prov
  - Contrangerea campului menu_id cu campul primary key ‘id’ de la menus_prov pentru dishes_prov -fk_dishes_prov_menus_prov
  - Contrangerea campului waiter_id cu campul primary key ‘id’ de la employees pentru orders_prov- fk_orders_prov_employees_prov
  - Contrangerea campului order_id cu campul primary key ‘id’ de la orders_prov pentru orders_drink_prov -fk_orders_drinks_prov_orders_prov
  - Contrangerea campului drink_id cu campul primary key ‘id’ de la drinks_prov pentru orders_drinks -fk_orders_drinks_prov_drinks_prov
  - Contrangerea campului order_id cu campul primary key ‘id’ de la orders_prov pentru chefs_orders_dishes_prov - fk_chefs_orders_dishes_prov_orders_prov
  - Contrangerea campului dish_id cu campul primary key ‘id’ de la dishes_prov pentru chefs_orders_dishes_prov - fk_chefs_orders_dishes_prov_dishes_prov
  - Contrangerea campului chef_id cu campul primary key ‘id’ de la employee pentru chefs_orders_dishes_prov - fk_chefs_orders_dishes_prov_employee_prov

De asemenea, pentru tabelele replicate am făcut un mecanism în pdb1 care să genereze id-uri începând cu 100 din 1 în 1, urmând a fi replicate, copiat și în pdb2. În cazul tabelei fragmentate cities avem o secvență care pornește de la 100 și generează id-uri din 2 în 2 că să fie doar pare iar pentru cea prov vom avea o secvență din 2 în 2 dar pornind de la 101 pentur numere impare pentru a avea o siguranță mai mare că nu se corup datele între fragmentări.