**УНИВЕРЗИТЕТ ФОН**

**Факултет за Информатика**



**Семинарска работа**

**Скопје,**

**Јуни, 2019 година**

**Семинарска работа**

**По предмет**

**Дистрибуирани комп. с-ми**

**Тема:**

**Фрагментација на дистрибуирани бази, вертикална, хоризонтална и хибридна**

Кандидат: Ментор:

Марио Илиќ д-рЛеонид Џиневски

Бр. на индекс 13647

Содржина

[**Листа на слики** 4](#_Toc12490890)

[**Abstract** 5](#_Toc12490891)

[**1.** **Вовед** 6](#_Toc12490892)

[**2.** **Вертикална фрагментација** 9](#_Toc12490893)

[**3.** **Хоризонтална фрагментација** 12](#_Toc12490894)

[**3.1.** **Примарна хоризонтална фрагментација** 13](#_Toc12490895)

[**3.2.** **Изведена хоризонтална фрагментација** 14](#_Toc12490896)

[**4.** **Хибридна фрагментација** 19](#_Toc12490897)

[**5.** **Насоки за вертикална фрагментација** 22](#_Toc12490898)

[**5.1.** **Групирање** 23](#_Toc12490899)

[**5.2.** **Разделување** 24](#_Toc12490900)

[**6.** **Вертикална фрагментација правила за корекција** 38](#_Toc12490901)

[**7.** **Упатства за генерирање хоризонтална фрагментација.** 39](#_Toc12490902)

[**7.1.** **Минималност и комплетност на хоризонталната фрагментација.** 40](#_Toc12490903)

[**8.** **Правила за точност на хоризонталната фрагментација** 43](#_Toc12490904)

[**9.** **Репликација** 45](#_Toc12490905)

[**10.** **Заклучок** 46](#_Toc12490906)

[**References** 47](#_Toc12490907)

# **Листа на слики**

[Слика 1 - Не фрагментирана верзија од ЕМР табела 9](#_Toc12489617)

[Слика 2 - EMP Sal\_Fragment 10](#_Toc12489618)

[Слика 3 - EMP\_NON\_Sal Fragment 10](#_Toc12489619)

[Слика 4 - MPLS\_EMPS fragment 14](#_Toc12489620)

[Слика 5 - LA\_EMPS fragment 14](#_Toc12489621)

[Слика 6 - NY\_EMPS fragment 14](#_Toc12489622)

[Слика 7 - Dept табела 15](#_Toc12489623)

[Слика 8 - MPLS\_DEPTS fragment 15](#_Toc12489624)

[Слика 9 - NY\_DEPTS fragment 15](#_Toc12489625)

[Слика 10 - LA\_DEPTS fragment 15](#_Toc12489626)

[Слика 11 - PROJ табела 16](#_Toc12489627)

[Слика 12 - PROJ1 табела 16](#_Toc12489628)

[Слика 13 - PROJ2 табела 16](#_Toc12489629)

[Слика 14 - DEPT4 17](#_Toc12489630)

[Слика 15 - DEPT5 18](#_Toc12489631)

[Слика 16 - PROJ 5 18](#_Toc12489632)

[Слика 17 - PROJ 6 18](#_Toc12489633)

[Слика 18 - Фрагменти од ЕМР табела 21](#_Toc12489634)

[Слика 19 - Единечна употреба на матрица за пр.6 25](#_Toc12489635)

[Слика 20 - Проширување на матрицата за користење 25](#_Toc12489636)

[Слика 21 - Пристапни фрекфенции за дистрибуиран систем 26](#_Toc12489637)

[Слика 22 - Process column affinit matrix 27](#_Toc12489638)

[Слика 23- Матрица на афинитет 27](#_Toc12489639)

[Слика 24 - Матрица на афинитет со пресметани дијагонални вредности 27](#_Toc12489640)

[Слика 25 - Пресметка на врската за C1 и C2 од пример 6 28](#_Toc12489641)

[Слика 26 - Врска за С0 30](#_Toc12489642)

[Слика 27 - Врска за С1 30](#_Toc12489643)

[Слика 28 - Врска за С1 30](#_Toc12489644)

[Слика 29 - Парцијално групирана афинитетна матрица 31](#_Toc12489645)

[Слика 30 - Групираната афинитетна матрица после преуредување на колоните 32](#_Toc12489646)

[Слика 31- Групираната афинитетна матрица после преуредување на редиците 32](#_Toc12489647)

[Слика 32 - Поделба на групираната афинитетна матрица 33](#_Toc12489648)

[Слика 33 - Стартни точки за поделбите на групираните афинитетни матрици 34](#_Toc12489649)

[Слика 34 - Матрица на втора точка на поделба 35](#_Toc12489650)

[Слика 35 - Матрица на третата точка на поделба 35](#_Toc12489651)

[Слика 36 - матрица за внатрешен блок на колони 36](#_Toc12489652)

[Слика 37 - Матрица со конечната поделба после SHIFT операцијата 36](#_Toc12489653)

[Слика 38 - PROJ табела 41](#_Toc12489654)

[Слика 39 - Фрагментација на PROJ табели базирана на минтерм предикати 43](#_Toc12489655)

# **Abstract**

Оваа семинарска работа како тема ја обработува фрагментацијата на дистрибуираните бази. На почетокот од семинарската работа опишано е што претставува фрагментацијата и наброени се техниките како: хоризонтална, вертикална и хибридна фрагментација. Понатаму секоја од овие техники е подетално опишана и објаснета со примери. Потоа прикажани се двата пристапи за дизајнирање на вертикална фрагментација: групирање и разделување.Следуваат правилата за корекција на VF како што се комплетност, реконструктивност и заедничкиот основен клуч. Дадени се упатства за генерирање хоризонтална фрагментација и правила за нејзината коректност. Како последна точка од семинарската работа претставува репликацијата на базите на податоци и нејзиното значење.

# **Вовед**

Фрагментацијата е функција на базата на податоци која ви овозможува на базата да контролира каде податоците се зачувани на табеларно ниво. Фрагментацијата овозможува да дефинирате групи на редови или клучеви за индекс во табела според некој алгоритам или шема. Секоја група или фрагмент (исто така наведена како партиција) може да ја зачувате во посебен dbspace поврзан со специфичен физички диск. Се користат SQL изјавите за да се креираат фрагментите и да се доделат на dbspaces.

Шемата што се користи за групирање на редови или клучеви за индекс во фрагменти се нарекува шема за дистрибуција. Дистрибутивната шема и множеството на dbspaces во кои се наоѓаат фрагментите заедно ја сочинуваат стратегијата за фрагментација. Одлуките што треба да се направат за да се формулира стратегија за фрагментација се објаснети во IBM-овиот Водич за изведба на IBM.

Откако ќе се одлучи дали ќе се фрагментираат редовите на табелата, индексните клучеви или двете и ќе се одлучи како редовите или клучевите треба да се дистрибуираат преку фрагменти, се одлучува за шема за спроведување на оваа дистрибуција. За опис на шемите за дистрибуција кои поддржуваат IBM® Informix® серверите за бази на податоци, се гледаат Дистрибутивни шеми за фрагментација на табелата.

Кога се креира фрагментирани табели и индекси, серверот со бази на податоци ја зачувува локацијата на секоја табела и индексниот фрагмент со други сродни информации во системската каталошка табела наречена sysfragments. Можете да се користите оваа табела за пристап до информации за нашите фрагментирани табели и индекси. Ако се користи кориснички дефинирана рутина како дел од изразот фрагментација, таа информација е снимена во sysfragexprudrdep. За опис на информациите што ги содржат овие табели на системските каталози, може да се погледне во IBM Informix Guide до SQL: Reference.

Од перспектива на крајниот корисник или на клиентска апликација, фрагментираната табела е идентична со нефрагментирана табела. Клиентските апликации не бараат никакви модификации за да им овозможат пристап до податоците во фрагментираните табели.

За некои шеми за дистрибуција, серверот за базата на податоци има информации за кои фрагменти ги содржат кои податоци, така што може да ги насочува клиентските барања за податоците до точниот фрагмент без пристап до ирелевантни фрагменти. (Серверот за базата на податоци не може да ги насочува клиентските барања за податоци до точниот фрагмент за круг-робин и некои шеми за дистрибуција базирани на експресија.) За повеќе информации, може да се види во Дистрибутивни шеми за фрагментација на табели.

Податоците може да се складираат на различни компјутери со фрагментирање на целата база на податоци во неколку парчиња наречени фрагменти. Секое парче се чува на друго место.

Фрагменти се логички единици на податоци складирани на различни локации во системот на дистрибуирани бази на податоци.

Пред да расправаме детално за фрагментацијата, набројуваме причини за фрагментирање на односот:

* **Употреба**

Општо земено, апликациите работат со погледи, а не со цели односи. Затоа, за дистрибуција на податоци, се чини дека е соодветно да се работи со подмножества на релацијата како единица на дистрибуција.

* **Ефикасност**

Податоците се чуваат близу до местото каде што најчесто се користи. Покрај тоа, податоците што не се потребни од "локални апликации" не се зачувани.

* **Паралелизам**

Со фрагменти како единица на дистрибуција, трансакцијата може да се подели на неколку под-барања кои работат на фрагменти. Ова треба да го зголеми степенот на конкурентност, или паралелизам, во системот, со што ќе овозможи трансакции кои можат да го сторат тоа безбедно да извршат паралелно.

* **Безбедност**

Податоците кои не се бараат од локалните апликации не се складираат и, следствено, не се достапни за неовластени корисници.

* **Недостатоци на фрагментацијата**

Фрагментацијата има две основни недостатоци, кои претходно ги споменавме:

* **Перформанси**

Ефикасноста на глобалната апликација која бара податоци од неколку фрагменти лоцирани на различни локации може да биде побавна.

* **Интегритет**

Контролата на интегритетот може да биде потешка ако податоците и функционалните зависности се фрагментирани и се наоѓаат на различни локации.

Фрагментираниот пристап или поделен фрагментациски пристап дели табелa во две или повеќе парчиња наречени фрагменти или партиции и овозможува складирање на овиe парчиња во различни локации.

Постојат три алтернативи на фрагментацијата:

• Вертикална фрагментација

• Хоризонтална фрагментација

• Хибридна фрагментација

Оваа дистрибутивна алтернатива се базира на верувањето дека не сите податоци во табелата се потребни на дадена локација. Освен тоа, фрагментацијата обезбедува зголемена паралелизам, пристап, катастрофи и безбедност / приватност. Во овој дизајн алтернатива, постои само една копија од секој фрагмент во системот (несложни фрагменти).

* **Делумна репликација**

 Во оваа дистрибутивна алтернатива, дизајнерот ќе направи копии на некои од табелите (или фрагменти) во базата на податоци и складирајте ги овие копии на различни локации. Ова се базира на верувањето дека фреквенцијата на пристап до табелите на бази на податоци  не се униформни. На пример, можеби Фрагмент 1 од EMP табелата може да биде почесто користен од Фрагмент 2 од табелата. За да се задоволи ова барање, дизајнерот може да одлучи да зачува само една копија од Фрагмент 2, но повеќе од една копија од Фрагмент 1 во системот. Повторно, зависи од бројот на потребните копии од Фрагмент 2, колку често се користат овие пристапни пребарувања и каде се генерираат овие пристапни пребарувања.

* ***Мешана дистрибуција***

Во оваа алтернатива на дизајнот, ние ја фрагментираме базата на податоци по желба, хоризонтално или вертикално, а потоа делумно реплицираме некои од фрагментите.

Фрагментацијата бара табелата да биде поделена на множество помали табели наречени фрагменти. Фрагментацијата може да биде хоризонтална, вертикална или хибридна (мешавина на хоризонтална и вертикална). Хоризонталната фрагментација понатаму може да се класифицира во две класи: примарна хоризонтална фрагментација (PHF) и изведена хоризонтална фрагментација (DHF). Кога размислуваат за фрагментацијата, дизајнерите треба да одлучат за степенот на грануларност за секој фрагмент. Со други зборови, колку од столбовите и / или редовите на табелата треба да бидат во фрагмент? Опсегот на опции е огромен. Од една страна, можеме да ги имаме сите редови и сите столбови на табелата во еден фрагмент. Ова очигледно ни дава неграгативна табела; житото е премногу груб ако планиравме да има барем еден фрагмент. Од друга страна, можеме да ставиме секоја ставка на податоци (вредност од една колона за еден ред) во посебен фрагмент. Ова зрно очигледно е премногу добро: би било тешко да се управува и би додало премногу надземни трошоци за обработка на прашања. Одговорот треба да биде некаде помеѓу овие две крајности. Како што ќе објасниме подоцна, оптималното решение зависи од типот и фреквенцијата на прашањата кои апликациите работат наспроти табелата. Во остатокот од овој дел, го истражуваме секој тип на фрагментација и го формализираме процесот на фрагментација.

# **Вертикална фрагментација**

Вертикалната фрагментација (VF) ги групира колоните на табелата во фрагменти. VF мора да се направи на таков начин што оригиналната табела може да се реконструира од фрагментите. Ова барање за фрагментација се нарекува "реконструкција". Ова барање се користи за реконструкција на оригиналната табела кога е потребно. Како резултат на тоа, секој VF фрагмент мора да ги содржи основните колони на табелата [1]. Бидејќи секој фрагмент содржи подмножество од вкупниот збир на колони во табелата, VF може да се користи за да се обезбеди сигурност и / или приватност на податоците. За да се создаде вертикален фрагмент од табела, се користи наредба за селектирање во која "Column\_list" е листа на колони од R што го вклучува примарен клуч.

Select Column\_list from R;

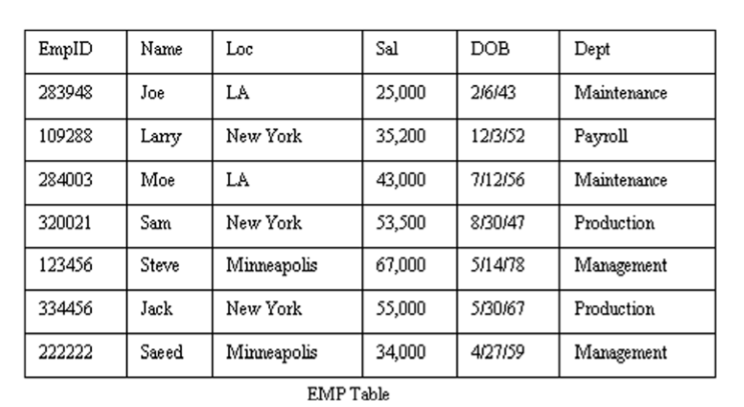
*Пример 1:*

Размислете за EMP табелата прикажана на слика 1. Да претпоставиме дека, поради безбедносни причини, информациите за плата за вработените треба да се одржуваат во серверот на седиштето на компанијата, кој се наоѓа во Минеаполис. За да се постигне ова, дизајнерот ќе ја фрагментира табелата вертикално во два фрагменти како што следи:

Create table EMP\_SAL as

Select EmpID, Sal

From EMP;



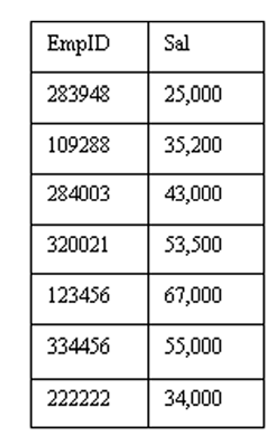
Слика 1 - Не фрагментирана верзија од ЕМР табела

Create table EMP\_NON\_SAL as

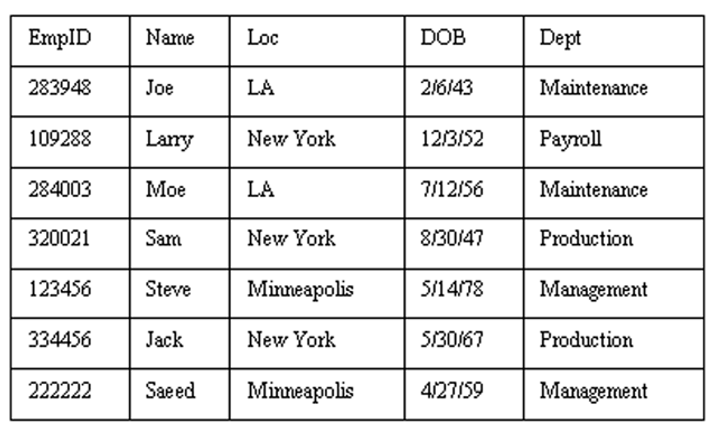
Select EmpID, Name, Loc, DOB, Dept

From EMP;

EMP\_SAL содржи информации за плата за сите вработени, додека EMP\_NON\_SAL ги содржи нечувствителните информации. Овие изјави генерираат вертикални фрагменти прикажани на Слика 2 и 3 од EMP табелата.По фрагментацијата, табелата EMP нема да се чува физички насекаде. Но, за да обезбедиме фрагментациона транспарентност - не барајќи корисниците да знаат дека табелата на ЕМП е фрагментирана - мора да бидеме во можност да ја реконструираме EMP табелата од VF фрагментите. Ова ќе им даде на корисниците илузија дека EMP табелата е зачувана непроменети. За да го направите ова, ние ќе ја користиме следнава изјава за придружба каде што е потребна EMP табелата:



Слика 2 - EMP Sal\_Fragment



Слика 3 - EMP\_NON\_Sal Fragment

Select EMP\_SAL.EmpID, Sal, Name, Loc, DOB, Dept

From EMP\_SAL, EMP\_NON\_SAL

Where EMP\_SAL.EmpID = EMP\_NON\_SAL.EmpID;

Притоа треба да се напомене дека оваа изјава за придружба може да се користи при дефинирање на приказ наречен "EMP" и / или може да се користи како онлајн-изглед во која било избрана изјава што ја користи виртуелната (физичкинепостоечки) табела "EMP."

# **Хоризонтална фрагментација**

Хоризонталната фрагментација (HF) може да се примени на табела или на фрагмент од табелата. Имајте на ум дека фрагментот од табелата е засебна табела. Затоа, во следнава дискусија кога го употребуваме терминoт табела, може да се повикаме на базна табела или фрагмент од табелата. HF ги групира редовите на табела врз основа на вредностите на една или повеќе колони [1]. Слично на вертикалната фрагментација, хоризонталната фрагментација мора да се направи на таков начин што основната табела може да се реконструира (реконструкција). Бидејќи секој фрагмент содржи подмножество од редовите во табелата, HF може да се користи за да се спроведе сигурност и / или приватност на податоците. Секој хоризонтален фрагмент мора да ги има сите колони од оригиналната база [3]. За да креирате хоризонтален фрагмент од табела, се користи изјава за избирање. На пример, следната изјава го избира редот од R што ја задоволува условот C:

Select \* from R where C;

Како што споменавме порано, постојат два пристапи за хоризонтална фрагментација. Една од нив се нарекува примарна хоризонтална фрагментација (PHF), а другата се нарекува изведена хоризонталнафрагментација (DHF).

## **Примарна хоризонтална фрагментација**

 Примарната хоризонтална фрагментација (PHF) партиционира табела хоризонтално врз основа на вредностите на една или повеќе колони од табелата.

Пример 2 го разгледува создавањето на три PHF фрагменти од EMP табелата врз основа на вредностите на колоната Loc.

*Пример 2:*

Размислете за EMP табелата прикажана на слика 1. Да претпоставиме дека имаме три експозитури, при што секој вработен работи само во една канцеларија. За полесно користење, одлучуваме дека информациите за одреден вработен треба да се чуваат на серверот DBMS во подрачната канцеларија каде што работи тој вработен. Затоа, EMP табелата треба да биде фрагментирана хоризонтално во три фрагменти врз основа на вредноста на колоната Loc какоприкажано подолу:

Create table MPLS\_EMPS as

Select \*

From EMP

Where Loc = ‘Minneapolis’;

Create table LA\_EMPS as

Select \*

From EMP

Where Loc = ‘LA’;

Create table NY\_EMPS as

Select \*

From EMP

Where Loc = ‘New York’;

Овој дизајн генерира три фрагменти, прикажани на сликите 4, 5 и 6. Секој фрагмент може да биде зачуван на серверот кој одговра на градот.

Повторно, по фрагментацијата, EMP табелата никаде нема да биде зачувана физички. За да се обезбеди транспарентност за хоризонталната фрагментација, треба да ја реконструираме EMP табелата од нејзините HF фрагменти. За да го сториме тоа, ја користиме следната union наредба секаде каде што се бара EMP табелата.

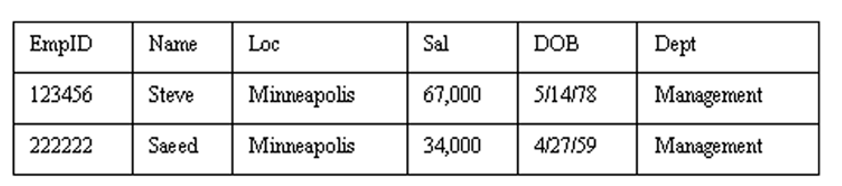
(Select \* from MPLS\_EMPS

Union

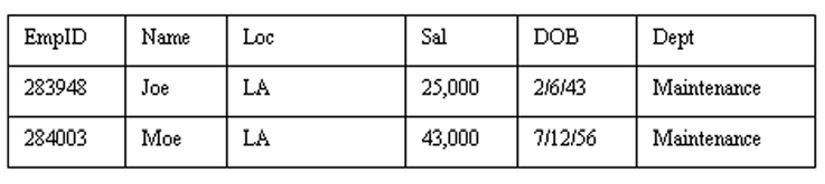
Select \* from LA\_EMPS)

Union

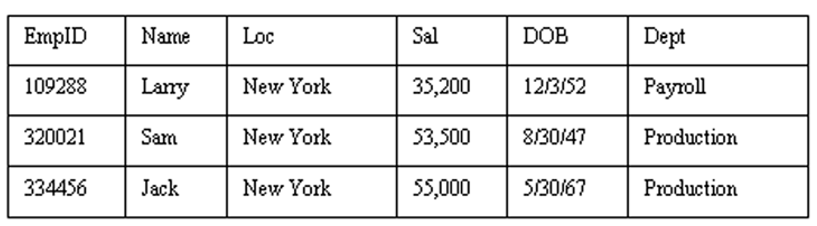
Select \* from NY\_EMPS;



Слика 4 - MPLS\_EMPS fragment



Слика 5 - LA\_EMPS fragment



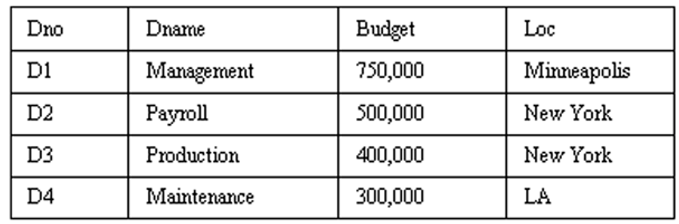
Слика 6 - NY\_EMPS fragment

## **Изведена хоризонтална фрагментација**

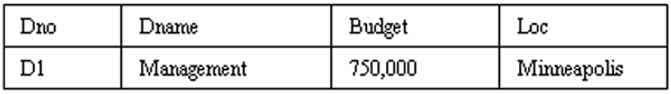
Наместо да користи PHF, дизајнерот може да одлучи да фрагментира табела според начинот на кој друга табела е фрагментирана. Овој тип фрагментација се нарекува изведена хоризонтална фрагментација (DHF). DHF обично се користи за две табели кои природно (и често) се приклучуваат. Затоа, складирањето на соодветните фрагменти од двете табели на истата локација ќе го забрза приклучувањето низ двете табели. Како резултат на тоа, имплицираниот услов за овој фрагментациски дизајн е присуството на колона за спојување низ двете табели.

*Пример 3:*

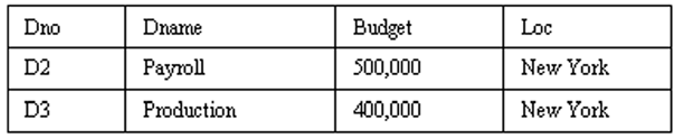
Слика 7 ја покажува табелата "DEPT (Dno, Dname, Budget, Loc)", каде Дно е примарен клуч на табелата. Да претпоставиме дека DEPT е фрагментиран врз основа на градот на одделот. Примена на PHF во табелата DEPT генерира три хоризонтални фрагменти, по еден за секој од градовите во базата на податоци, како што е прикажано на сликите 8, 9, 10.Сега, да ја разгледаме табелата "PROЈ", како што е прикажано на Сликa 11. Ние можеме да ја поделиме табелата на PROJ врз основа на вредностите на Dno колоната во фрагментите на табелата DEPT со следниве SQL наредби. Овие наредби ќе произведат изведени фрагменти од табелата на PROJ како што е прикажано на Слика 12, 13. Забележете дека во PROJ3 нема редови,бидејќи одделот "Д4" не управува со ниеден проект.



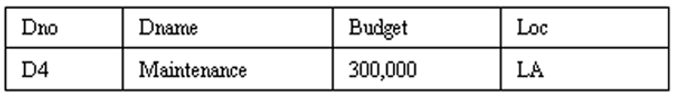
Слика 7 - Dept табела



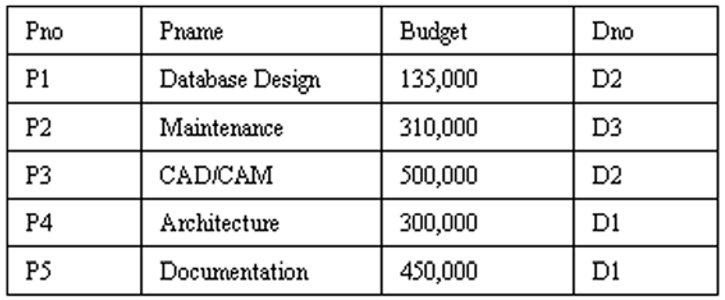
Слика 8 - MPLS\_DEPTS fragment



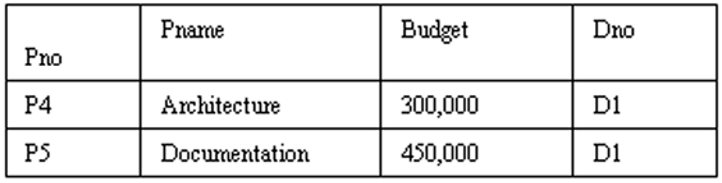
Слика 9 - NY\_DEPTS fragment



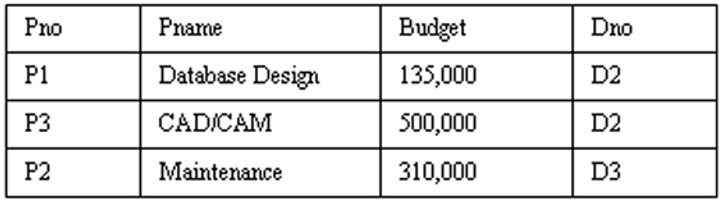
Слика 10 - LA\_DEPTS fragment



Слика 11 - PROJ табела



Слика 12 - PROJ1 табела



Слика 13 - PROJ2 табела

Create table PROJ1 as

Select Pno, Pname, Budget, PROJ.Dno

From PROJ, MPLS\_DEPTS

Where PROJ.Dno = MPLS\_DEPTS.Dno;

Create table PROJ2 as

Select Pno, Pname, Budget, PROJ.Dno

From PROJ, NY\_DEPTS

Where PROJ.Dno = NY\_DEPTS.Dno;

Create table PROJ3 as

Select Pno, Pname, Budget, PROJ.Dno

From PROJ, LA\_DEPTS

Where PROJ.Dno = LA\_DEPTS.Dno;

Треба да биде прилично очигледно дека сите редови во PROJ1 имаат соодветни редови во фрагментот MPLS\_DEPTS, и на сличен начин, сите редови во PROJ2 имаат соодветните редови во фрагментот NY\_DEPTS. Складирањето на изведениот фрагмент на истиот сервер за бази на податоци каде што е изведениот фрагмент, ќе резултира со подобри перформанси, бидејќи сите приклучоци низ фрагментите на двете табели ќе резултираат со сооднос на хит од 100% (сите редови во еден фрагмент имаат соодветни редови во другиот) .

*Пример 4:*

За овој пример, претпоставиме дека понекогаш сакаме да ги најдеме оние проекти со кои раководат одделенијата кои имаат буџет помал или еднаков на 500.000 (буџет на одделот, а не буџет на проектот), а во други случаи сакаме да ги најдеме тие проекти кои се управувани од одделите кои имаат буџет од повеќе од500.000. За да го постигнеме ова, ние фрагментираме DEPT врз основа на буџетот на одделот. Сите одделенија со буџет помал или еднаков на 500.000 се зачувани во DEPT4, а другите сектори се складирани во фрагментот DEPT5. Сликите 14 и 15 покажуваат DEPT4 и DEPT5, соодветно.

За лесно да одговориме на типот на прашања што ги издвоивме во овој пример, треба да создадеме два изведени хоризонтални фрагменти од PROJ табелата врз основа на DEPT4 и DEPT5 како што е прикажано подолу.

Create table PROJ5 as

Select Pno, Pname, Budget, PROJ.Dno

From PROJ, DEPT4

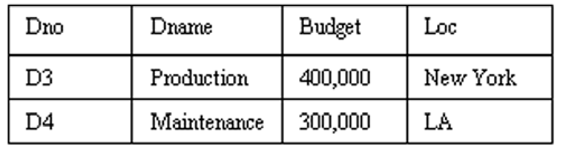
Where PROJ.Dno = DEPT4.Dno;

Create table PROJ6 as

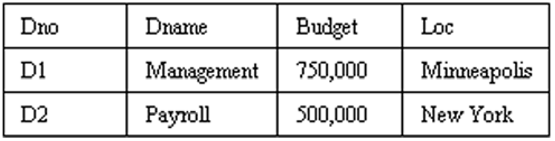
Select Pno, Pname, Budget, PROJ.Dno

From PROJ, DEPT5

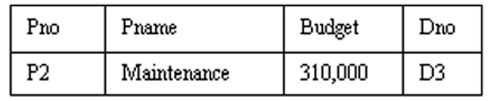
Where PROJ.Dno = DEPT5.Dno;



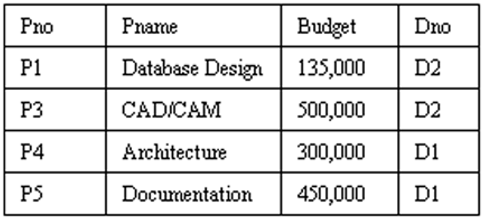
Слика 14 - DEPT4



Слика 15 - DEPT5



Слика 16 - PROJ 5



Слика 17 - PROJ 6

# **Хибридна фрагментација**

Хибридната фрагментација (HyF) користи комбинација на хоризонтална и вертикална фрагментација за генерирање на фрагментите што ни се потребни. Постојат два пристапа за тоа. Во првиот пристап, ние генерираме група хоризонтални фрагменти, а потоа вертикално фрагментираме еден од повеќе од овие хоризонтални фрагменти. Во вториот пристап, ние генерираме група вертикални фрагменти, а потоа хоризонтално фрагментираме еден или повеќе од овие вертикални фрагменти. Во секој случај, финалните фрагменти произведени се исти [2]. Овој пристап за фрагментација обезбедува најмногу флексибилност за дизајнерите, но во исто време тоа е најскапиот пристап во однос на реконструкцијата на оригиналната табела.

*Пример 5:*

Да претпоставиме дека информациите за плата на вработените треба да се одржуваат во посебен фрагмент од несаканите информации како што е дискутирано погоре. Вертикалниот фрагментациски план ќе генерира вертикални фрагменти EMP\_SAL и EMP\_NON\_SAL како што е објаснето во Пример 1. Несаканите информации треба да бидат фрагментирани во хоризонтални фрагменти, каде што секој фрагмент содржи само редови кои се совпаѓаат со градот каде што работат вработените. Ова можеме да го постигнеме со примена на хоризонтална фрагментација на EMP\_NON\_SAL фрагментот од EMP табелата. Следниве три SQL изјавите покажуваат како се постигнува ова.

Create table NON\_SAL\_MPLS\_EMPS as

Select \*

From EMP\_NON\_SAL

Where Loc = ‘Minneapolis’;

Create table NON\_SAL\_LA\_EMPS as

Select \*

From EMP\_NON\_SAL

Where Loc = ‘LA’;

Create table NON\_SAL\_NY\_EMPS as

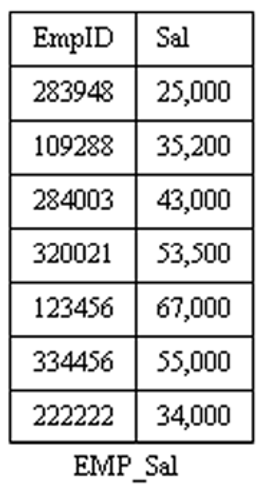
Select \*

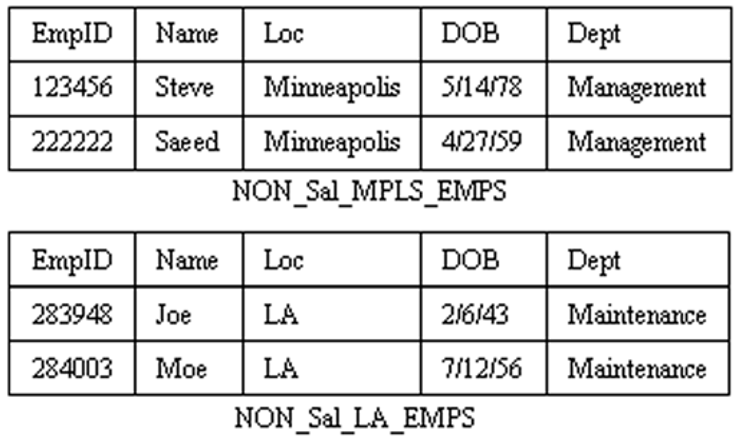
From EMP\_NON\_SAL

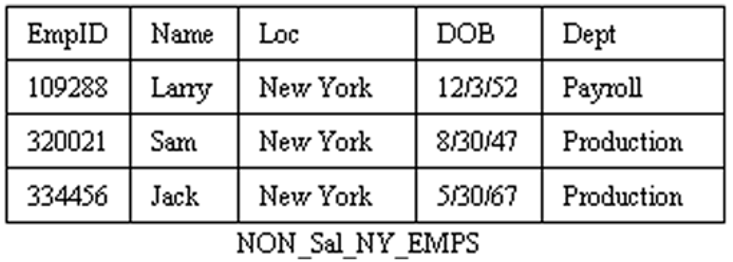
Where Loc = ‘New York’;

Конечната дистрибуирана база на податоци е прикажана на Слика 2.10.

Набљудување: Привремениот EMP\_NON\_SAL фрагмент не е физички зачуван насекаде во системот откако е хоризонтално фрагментиран. Како резултат на тоа, може да се заобиколи генерирање на овој фрагмент со користење на следниве множества на SQL извештаи за генерирање на бараните фрагменти директно од EMP табелата.







Слика 18 - Фрагменти од ЕМР табела

Create table NON\_SAL\_MPLS\_EMPS as

Select EmpID, Name, Loc, DOB, Dept

From EMP

Where Loc = ‘Minneapolis’;

Create table NON\_SAL\_LA\_EMPS as

Select EmpID, Name, Loc, DOB, Dept

From EMP

Where Loc = ‘LA’;

Create table NON\_SAL\_NY\_EMPS as

Select EmpID, Name, Loc, DOB, Dept

From EMP

Where Loc = ‘New York’;

# **Насоки за вертикална фрагментација**

Постојат два пристапи за дизајнирање на вертикална фрагментација и предложени поделби во литературата [Hoffer75] [Hammer79] [Sacca85]. Во остатокот од овој дел, ние прво ќе обезбедиме преглед на овие две опции, а потоа презентираме повеќе детали за опцијата на разделување.

## **Групирање**

Групирањето е пристап кој започнува со создавање на што е можно повеќе вертикални фрагменти, а потоа постепено намалување на бројот на фрагменти со спојување на фрагментите заедно. Првично, ние создаваме еден фрагмент по колона без клуч, поставувајќи ја колоната на невезените и примарниот клуч на табелата во секој вертикален фрагмент.Овој прв чекор создава толку вертикални фрагменти како бројот на колони без клуч во табелата. Најчесто, овој степен на фрагментација е премногу фин и непрактичен. Пристапот за групирање користи приклучоци преку примарниот клуч, да ги групира некои од овие фрагменти заедно, и ние го продолжуваме овој процес додека не се постигне саканиот дизајн. Настрана од потребата да се исполнат барањата за една или повеќе апликации, има многу малку ограничувања врз групите (фрагменти) што ги создаваме во овој пристап. На пример, истата колона од колона може да учествува во повеќе од една група- односно, групите може да имаат преклопувачки (невики) колони. Ако се случи ова "преклопување", очигледно, ќе се додаде во надземната контрола на репликацијата во дистрибуиран систем на DBMS. Како резултат на тоа, групирањето обично не се смета за валиден дизајнирачки пристап за вертикална фрагментација. За повеќе детали за групирање видете [Hammer79] и [Sacca85]. Hammer и Niamir воведоа групирање за централизирани DBMS и Sacca и Wiederhold дискутираа за групирање за дистрибуирани DBMS.

## **Разделување**

Раздвојувањето е во суштина спротивно на групите. Во овој пристап,табелата е фрагментирана со ставање на секоја колона без клуч во еден (и само еден) фрагмент, фокусирајќи се на идентификување на множество на потребни колони за секој вертикален фрагмент. Како таков, не постои преклопување на столбовите на непримарните клучеви во вертикалните фрагменти кои се создаваат со користење на разделување. Хофер и Северанс [Hoffer75] прв пат претставија разделување за централизирани системи, додека Навате и колегите [Navathe84] воведоа разделување за дистрибуирани системи. Постои генерален консензус дека наоѓањето на идеален вертикален фрагментациски дизајн - оној што ги задоволува барањата за голем сет на апликации - не е изводливо. Во идеален дизајн на вертикален фрагмент, секоја апликација треба само да пристапи до колоните во еден вертикален фрагмент. Ако одредени групи на колони секогаш се обработуваат заедно со апликацијата, процесот што се користи за создавање на овој дизајн е тривијален. Но, реалните апликации не се однесуваат секогаш како што сакаме. Оттука, за база на податоци што содржи многу табели со многу колони, треба да развиеме системски пристап за дефинирање на нашата вертикална фрагментација. Како што е истакнато од Хамер и Ниамир [Hammer79], постои директна корелација помеѓу бројот на колони во табелата и бројот на можни вертикалниопции за фрагментација. Табелата со m колони може да биде вертикално поделена на B (m) различни алтернативи, каде што B (m) е m-от Бел број. За големи m, B (m) се приближува mm. На пример, ако табелата има 15 колони, тогаш бројот на можни вертикални фрагменти е 109 и бројот на вертикални фрагменти за маса со30 колони е 1023. Очигледно, проценката на 109 (или 1.000.000.000) алтернативи за маса од 15 колони не е практична. Наместо да ги евалуираат сите можни вертикални фрагменти, дизајнерите можат да го користат метричкиот афинитет или блискоста на колоните еден до друг за да одлучуваат без разлика дали групата колони треба да биде ставена во истиот фрагмент. Афинитетот на колоните го изразува степенот до кој тие се користат заедно за обработка. Со комбинирање на фреквенцијата на пристап до колоните на табелата со примената на примената на овие апликации, може да се создаде матрица на афинитет што претставува основа за вертикално фрагментирање на табелата.

* **Разделување во дистрибуираните системи**

Во овој дел, ќе го претставиме предлогот на Navathe и неговите колеги [Navathe84] во врска со еден пример. За повеќе детали за пристапот, читателот треба да ја види референтната публикација.

*Пример 6:*  
  Размислете за апликациите "AP1", "AP2", "AP3" и "AP4" како што е прикажано. Овие апликации работат на табелата "T" дефинирана како "T (C, C1, C2, C3, C4)", каде што C е главниот столб на табелата.

AP1: Select C1 from T where C4 = 100;

AP2: Select C4 from T;

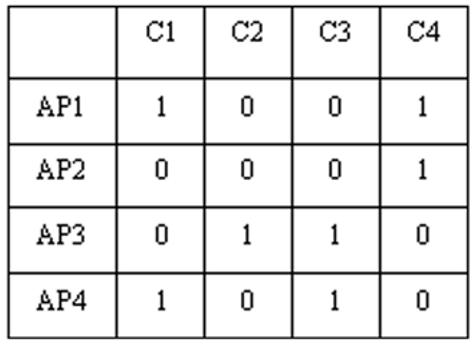
AP3: Update T set C3 = 15 where C2 = 50;

AP4: Update T set C1 = 5 where C3 = 10;

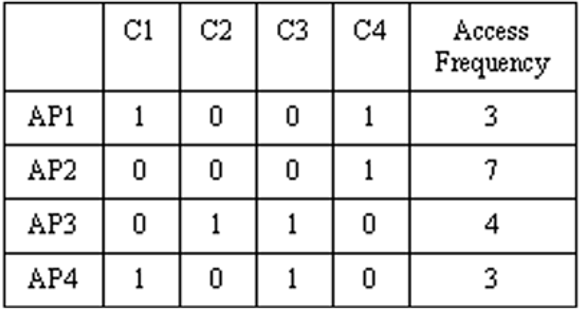
* **Матрица за користење**

 За систем со еден сајт, овие апликации се локални и ќе имаат матрица за користење прикажана на Слика 19. Како што може да се види, матрицата за користење е дводимензионална матрица која покажува дали атрибут (колона) се користи од некоја апликација или не. Ќелијата во позиција (APi, Cj) е поставена на 1 ако апликацијата "APi" пристапува до колоната "Cj", во спротивно е поставена на 0.

*Набљудување 1*: Матрицата за користење само покажува дали колона се користи од некоја апликација. Сепак, матрицата не покажува колку пати апликацијата пристапува до колоните на табелата во одреден временски период. Фреквенцијата на пристап е термин кој претставува колку пати апликацијата работи во одреден временски период. Периодот на ова мерење може да биде еден час, еден ден, една недела, еден месец и така натаму - како што одлучи дизајнерот. Слика 20 го отсликува проширувањето на матрицата за користење за да ја вклучи фреквенцијата за пристап на секоја апликација.



Слика 19 - Единечна употреба на матрица за пр.6



Слика 20 - Проширување на матрицата за користење

*Набљудување 2:* Ниту матрицата за употреба ниту фреквенциите за пристап немаат индикации за дистрибуција. Меѓутоа, во дистрибуиран систем апликацијата може да има различни фреквенции на различни локации. На пример, во четири-системски систем, AP2 може да работи четири пати на S2 и три пати на S3. Исто така е можно дека AP2 може да работи седум пати на местото S2 и нула пати на друго место. Во двата случаи, фреквенцијата сè уште ќе биде прикажана како седум во матрицата за употреба. Исто така, секој пат кога апликацијата работи на некоја локација, може да направи повеќе од еден пристап до табелата (и нејзините колони). На пример, да претпоставиме дека имавме друга апликација, AP5, дефинирана како што следува:

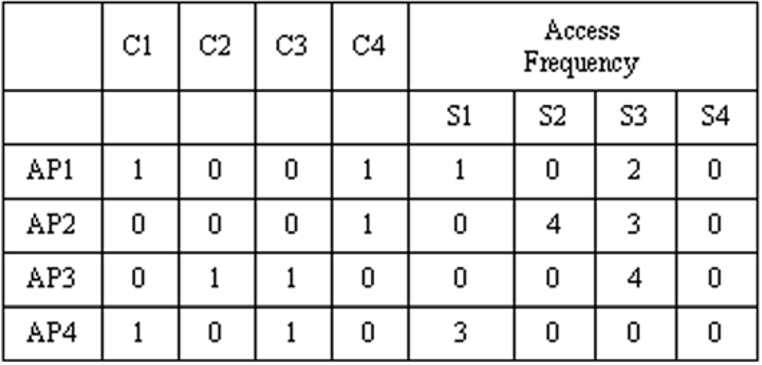
Begin AP5

Select C1 from T where C4 = 100;

Select C4 from T;

End AP1;

Во овој случај, AP5 прави две референци на T секој пат кога работи. Како резултат на тоа, вистинската фреквенција на пристап за AP5 се пресметува како "ACC (Pi) \* REF (Pi)", каде што ACC (Pi) е бројот на времетраењето на апликацијата и REF (Pi) е бројот на пристапи Pi направи до Т секој пат кога работи. За да се поедностави дискусијата, претпоставуваме "REF (Pi) = 1" за сите процеси, што резултира со "ACC (Pi) \* REF (Pi) = ACC (Pi)." Ако ги вклучиме фреквенциите за пристап на апликациите во секоја сајт за нашиот оригинален пример (без AP5), ова ја прави матрицата за користење тридимензионална матрица, каде што во третата оска ја одржуваме фреквенцијата на секоја апликација за секоја локација. Бидејќи таквата репрезентација е тешко да се прикаже на дводимензионална хартија, ја израмнуваме матрицата како што е прикажано на Слика 21.

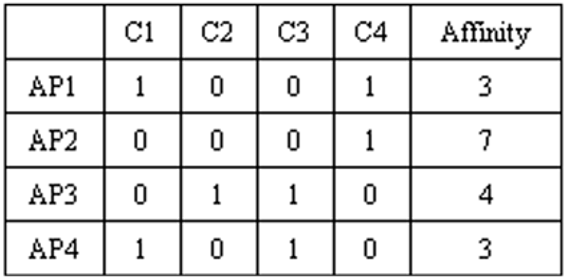


Слика 21 - Пристапни фрекфенции за дистрибуиран систем

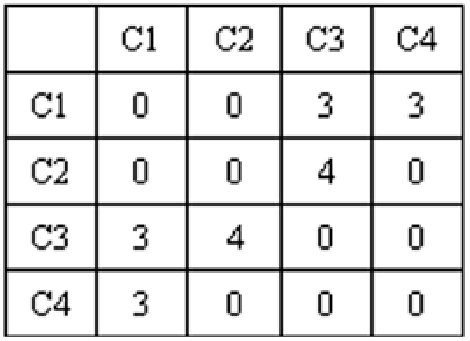
Со додавање на фреквенции за пристап за секоја апликација на сите веб-страници, можеме да добиеме афинитет или блискост што секоја колона ја има со другите колони референцирани од истата апликација. Оваа матрица се нарекува матрица за афинитет на процесот колона, и таа е прикажана за овој пример на слика 22

Првиот ред во оваа матрица покажува афинитет на C1 до C4 (или C4 до C1) за AP1 е 3. Слично на тоа, третиот ред укажува на афинитетот на C2 до C3 (или C3 до C2) за AP3 да биде 4; C1 до C3 (или C3 до C1) за AP4 да биде 3; и така натаму. Забележете дека бидејќи AP2 користи само C4, за AP2 нема афинитет меѓу C4 и другите колони. Со цел да ги собереме овие афинитети, ние можеме да ги отстраниме имињата на апликациите од матрицата и да создадеме дводимензионална матрица како што е прикажано на Слика 23. Оваа матрица го покажува афинитетот на секоја колона од табелата со други колони, без оглед на апликациите што ги користат. Ова го нарекуваме матрица за афинитет.

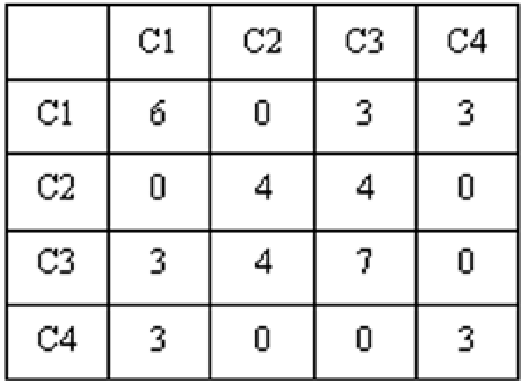
Забележете на слика 23 дека дијагоналните клетки-клетки во позиција "(i, i)" за "i = 1 до 4" имаат вредност 0. Бидејќи дијагоналните клетки не зачувуваат никакви вредности, можеме да го пресметаме афинитетот на колоната Ci во однос на сите други колони и ја чува оваа вредност во "Cell (i, i)" во матрицата. C1 има афинитет со C3 пондериран на 3 и афинитет со C4 пондериран на 3. Затоа, афинитетот на C1 низ сите апликации на сите локации се наоѓа со собирање на неговите тежини на афинитет со сите други колони "3 + 3 = 6. "Ние ќе ја внесеме оваа вкупна вредност на афинитет во" Cell (1,1). "Слично на тоа, C3 има две афинитети - оној со C1 пондериран на 3 и еден со C2 пондериран во 4. C3 ја има вкупната афинитетна вредност на



Слика 22 - Process column affinit matrix



Слика 23- Матрица на афинитет



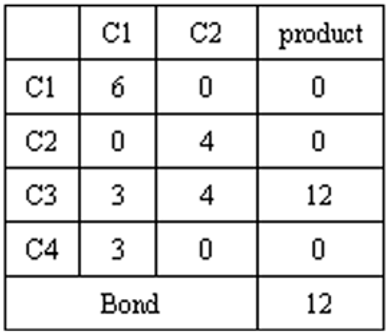
Слика 24 - Матрица на афинитет со пресметани дијагонални вредности

"3 + 4 = 7" за сите апликации на сите сајтови, кои ги вметнуваме во "Cell (3,3)." Со додавање на сите афинитети на секоја колона (сумирање низ редовите), може да се пресмета вкупниот афинитет за секоја колона , чувајќи ги во дијагоналните клетки. Слика 24 ги прикажува резултатите. Во следниот дел, ќе разгледаме алгоритам кој генерира наш вертикален фрагментациски дизајн врз основа на информациите за афинитет содржани во оваа матрица.

* **Алгоритам за енергетска врска на Бонд**

Информациите во матрицата за афинитет прикажани на слика 24 може да се користат за групирање на колоните од Т заедно. McCormick и колегите предлагаат користење на Алгоритмот за енергија на Бонд (BEA) [McCormick72] за ова. Овој алгоритам ја зема матрицата за афинитет како влезен параметар и генерира нова матрица наречена кластерирана афинитетна матрица како нејзин излез. Класифицираната афинитетна матрица е матрица која ги редефинира колоните и редовите на афинитетната матрица, така што колоните со најголем афинитет едни за други се "групирани заедно" во истиот кластер-кој потоа се користи како основа за разделување на нашата табела во вертикални фрагменти. Деталите за овој алгоритам се надвор од опсегот на ова поглавје. Во она што следува, ние само ќе разговараме за чекорите во алгоритмот за нашиот пример.

*Чекор 1:* поставување на првите две колони. Во овој чекор, ги ставаме првите две колони од матрицата за афинитет во нова матрица (кластерска афинитетна матрица). Во нашиот пример, првите две колони се C1 и C2. Според McCormick и колегите, енергијата на врската на било кои две колони како што се "X1" и "X2" - прикажана како "Бонд (X1, X2)" - е редуцирана сума на производот од вредностите на афинитет заX1 и X2, земени од афинитетната матрица.



Слика 25 - Пресметка на врската за C1 и C2 од пример 6

Сликата 25 ја покажува калкулацијата на енергетската врска за C1 и C2 од дадените примери. Во оваа фигура/бројка креиравме нова матрица со тоа што ги копиравме првите две колони од афинитетната матрица. Следно, ја зголемивме оваа нова матрица со додавање нова колона "производ", каде што го покажуваме резултатот од производот (множење) на двете вредности во истиот ред, лево од колоната. На пример, производот за редот "C3" во оваа фигура/бројка е 12, бидејќи "3 \* 4 = 12." Конечно, го зголемивме повторно со вклучување на редот за “Врска“кој едноставно го зачувува збирот на вредностите од сите колони на продуктот.

*Чекор 2:.* Поставеност на останати колони. Во овој чекор, го ставаме секоја од останатите колони (C3 и C4 во нашиот пример), една по една, во новогрупираната афинитетна матрица-што ние всушност не сме ја прикажале, но бидејќи содржи само C1 и C2, можеме да разбереме како изгледа без да го претставиме во посебна фигура/бројка. Прво, ќе го ставиме C3, а потоа и C4, во новата матрица. Целта на овој чекор е да се утврди каде треба да се стави секоја "нова" колона. Во примерот, прво треба да одлучиме "каде" да ја додадеме колоната С3 од матрицата за афинитет на матрицата која веќе содржи C1 и C2. За C3, постојат три опци - да ги поставиме C3 информациите за афинитет на левата страна од C1 (што ја прави најлевата колона во новата матрица); да ги поставиместавиме C3 информациите помеѓу C1 и C2, или да ги поставиме C3 информациите за афинитет на десната страна на C2 (што ја прави најдесната колона во новата табела). Потоа треба да ги пресметаме вредностите на придонесот на сите можни поставености на C3 афинитетните информации, користејќи ја формулата дадена во равенка 1. Откако ќе се пресметаат сите вредности на придонесите, ќе ја избереме поставеноста што обезбедува највисок придонес во мерката на афинитет.

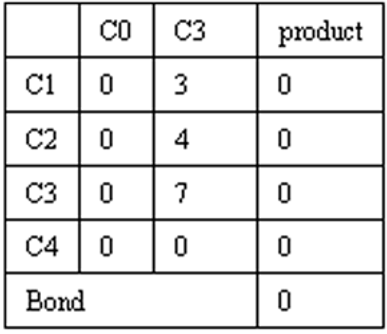
* **Пресметка на придонес за врска.**

Cont(X1, X2, X3) = 2 ∗ Bond(X1, X2) + 2 ∗ Bond(X2, X3) −

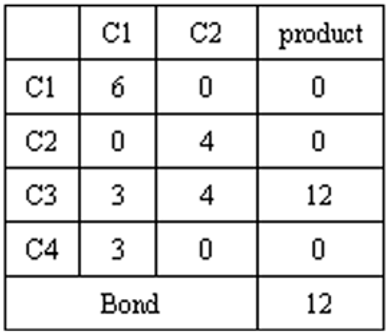
2 ∗ Bond(X1, X3) *Равенка 1.*

Општо земено, при додавање на колона на афинитетна матрица во групирана афинитетна матрица, мораме да ги разгледаме двата гранични услови - најлевите и најблиски позиции за новата колона. "X2" во формулата за придонес се заменува со колоната што се разгледува, додека "X1" е колоната од левата страна, а "X3" е колоната од десната страна. Меѓутоа, кога ќе го земеме во предвид поставувањето на новата колона во најлевата позиција, нема колони лево од тоа - затоа, користиме псевдоколона со име "C0" за да ја претставиме оваа непостоечка колона од лево. Слично на тоа, кога ќе го земеме во предвид поставувањето на новата колона во најдсната положба, употребуваме друга псевдоколона наречена "Cn" за да ја претстави непостоечката колона од своја страна. Во нашите пресметки, и C0 и Cn имаат афинитет на нула. Затоа, кога ќе размислиме за додавање на C3 во групата на афинитети матрици за примерот, постојат три опции: "(C3, C1, C2)", "(C1, C3, C2)" или "(C1, C2, C3) ). "За да се утврди која опција дава највисока вредност за мерката на афинитет, мора да се пресмете придонесот на C3 кој се додава лево од C1, помеѓу C1 и C2 и десно од C2. Така што треба да се пресмета: *“Cont(C0,C3, C1)”, “Cont(C1, C3, C2)”, и “Cont(C2, C3, Cn)”* и да се споредат резултатите. За да се пресмета придонесот на "(C0, C3, C1)," треба да ги разбереме врските за "(C0, C3)," "(C3, C1)," и "(C0, C1)". Откако ќе ги одредиме овие врски, можеме да го користиме равенството 1 за да пресметаме "Cont (C0, C3, C1) = 2 \* Bond (C0, C3) + 2 \* Bond (C3, C1) -2 \* Bond (C0, C1) . "Сликите 26, 27, и 28 ги прикажуваат овие пресметки на врската. Како што покажуваат овие бројки, функцијата Бонд секогаш ќе се врати на нула кога или C0 или Cn е предадена на него како параметар, бидејќи C0 и Cnalways имаат афинитет на нула.

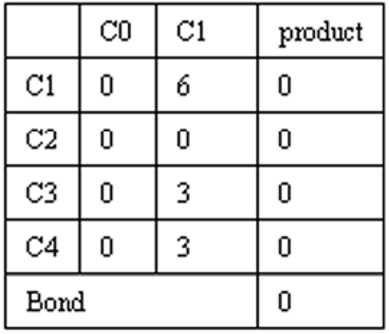
|  |  |
| --- | --- |
|  |  |



Слика 26 - Врска за С0



Слика 27 - Врска за С1



Слика 28 - Врска за С1

Со заменување на функциите Бонд со нивните вредности во формулата за придонес, го пресметуваме придонесот на оваа подреденост како "Cont (C0, C3, C1) = 2 \* 0 + 2 \* 39 - 2 \* 0 = 78." Слично на тоа, се пресметуваат придонесите на подреденостите "(C1, C3, C2)" и "(C1, C2, C3)" како:

For ordering (C1, C3, C2):

Bond(C1, C3) = 39

Bond(C3, C2) = 44

Bond(C1, C2) = 12

Cont(C1, C3, C2) = 2\*39 + 2\*44 - 2\*12 = 142

For ordering (C2, C3, Cn):

Bond(C2, C3) = 44

Bond(C3, Cn) = 0

Bond(C2, Cn) = 0

Cont(C1, C2, C3) = 2\*44 + 2\*0 - 2\*0 = 88

Врз основа на овие пресметки, подреденоста "(C1, C3, C2)" обезбедува највисок придонес. Користејќи го ова подредување, Слика 2.19 ја покажува тековната дефиниција на новата групирана афинитетна матрица. Бидејќи оваа матрица сè уште нема С4, таа се нарекува делумна афинитетна матрица на групи. По додавањето на С3 во матрицата, таа е подготвена да и се додадат информациите за афинитет за С4 како последна колона. Додавањето C4 во матрицата бара пресметување на придонесите од следните подредености "(C0, C4, C1)," "(C1, C4, C3)," "(C3, C4, C2)," и "(C2, C4, Cn)". Следните се обврзници и придонеси од горните подредености опции:

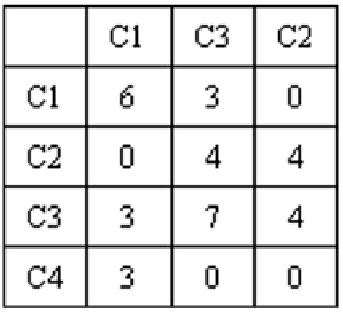
For ordering (C0, C4, C1):

Bond(C0, C4) = 0

Bond (C4, C1) = 27

Bond(C0, C1) = 0

Cont(C0, C4, C1) = 2\*0 + 2\*27 – 2 \*0 = 54



Слика 29 - Парцијално групирана афинитетна матрица

For ordering (C1, C4, C3):

Bond(C1, C4) = 27

Bond(C4, C3) = 9

Bond(C1, C3) = 39

Cont(C1, C4, C3) = 2\*27 + 2\*9 – 2\*39 = –6

For ordering (C3, C4, C2):

Bond(C3, C4) = 9

Bond(C4, C2) = 0

Bond(C3, C2) = 44

Cont((C3, C4, C2) = 2\*9 + 2\*0 – 2\*44 = –70

For ordering (C2, C4, Cn):

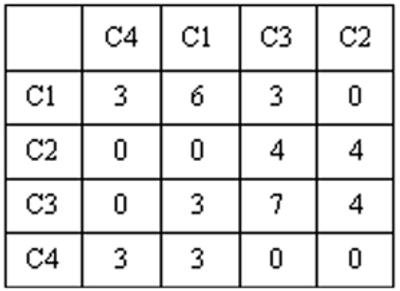
Bond(C2, C4) = 0

Bond(C4, Cn) = 0

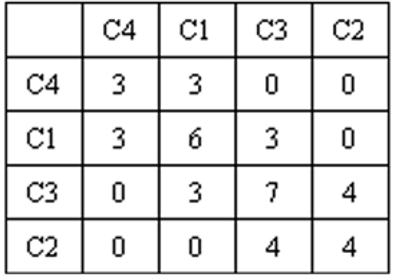
Bond(C2, Cn) = 0

Cont((C1, C4, C2) = 2\*0 + 2\*0 - 2\*0 = 0

Од овие пресметки на придонесите, подреденоста "(C0, C4, C1)" дава највисок придонес и затоа е избрана. Слика 30 го прикажува конечното подредување на информациите за афинитет за столбовите од табелата "Т" во групираната афинитетна матрица. Откако колоните ќе бидат ставени во правилен редослед, редовите се подредуваат слично, завршувајќи со конечната групирана афинитетна матрица како што е прикажано на Слика 31.



Слика 30 - Групираната афинитетна матрица после преуредување на колоните

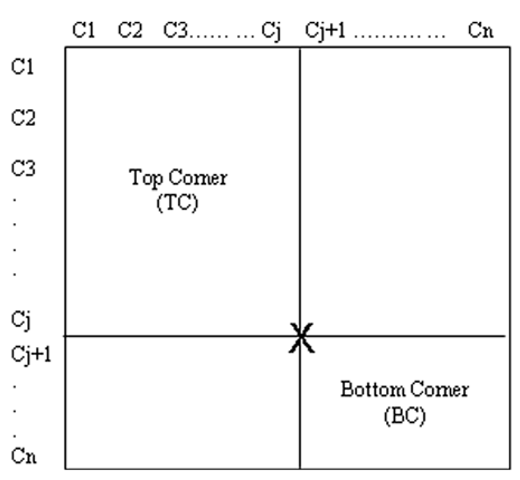


Слика 31- Групираната афинитетна матрица после преуредување на редиците

*Чекор 3: Поделба на табелата во вертикални фрагменти.* Третиот чекор е последниот во процесот. Во овој чекор, треба да ја поделиме групираната афинитетна матрица во горниот дел и долниот дел како што е прикажано на Слика 32. Меѓу сите апликации што пристапуваат до оваа табела, некои пристапуваат само до колоните во горниот агол (TC), некои пристапуваат само до колоните во долниот агол (BC), а останатите колони имаат пристап во двата агли (BOC). Сметаме (n-1) можни локации на X-точката долж дијагоналата, каде што **n** е големината на матрицата (исто така, бројот на не-клучните колони на табелата). Не-преклопувачка партиција се добива со избирање на X така што целната функција "Z" како што е дефинирана подолу е максимизирана.

Goal Function: Z = TCW \* BCW – BOCW2

Во Z, TCW е вкупниот број на влезови на апликации во колоните во TC (само колони од горниот агол), BCW е вкупниот број влезови на апликации во колони во BC (само колони од долниот агол) и BOCW е вкупниот број влезови на апликации во колони од двата квадранта (мора да се пристапи во најмалку една колона од TC и најмалку една колона од BC). Поделбата што одговара на максималната вредност на Z е прифатена ако Z е позитивна и е одбиена поинаку. Оваа формула е заснована на верувањето дека "добрата" стратегија за поделба ги зголемува вредностите на TCW и BCW додека ја намалува вредноста на BOCW. Со други зборови, влезот се обидува да го максимизира производот TCW \* BCW. Ова, исто така, резултира со избор на вредности за TCW и BCW кои се блиску до еднакви колку што е можно. Така, функцијата ќе произведе фрагменти кои се "избалансирани". За да се најде поделбата што дава најдобрa поделба на колоните, прво треба да ги поделиме колоните во "една колона п.н.е." и "n-1 колона TC". Се пресметува вредноста на Z за оваа почетна точка, а потоа постојано се додаваат колони од TC до BC, додека TC не остане само со една колона. Од овие пресметки, се бира поделбата која има највисока вредност Z. Се отповикува афинитетната матрицата за овој пример (прикажан тука повторно на Слика 33).



Слика 32 - Поделба на групираната афинитетна матрица

Како што е прикажано на Слика 34, започнуваме да ја делиме табелата со ставање C4, C1 и C3 во TC и C2 во BC. На оваа слика, "иконата Х" го означува горниот лев агол на BC и долниот десен агол на TC.

Пресметката TC ќе ја вклучува вредноста на афинитет за сите апликации кои пристапуваат до една од TK колоните (C4, C1 или C3), но не пристапуваат до било какви BC колони (C2). Во овој случај, апликациите AP1, AP2 и AP4 пристапуваат до TC-само колони, AP3 пристапува и до TC и BC колони, и ниедна апликација не е само BC. Се користат функцијата "AFF (a)" за да претставува афинитет за апликација "a". Со други зборови, во примерот, ова е кратка ознака за гледање на вредноста за колоната за афинитети на Слика 33 за редот што одговара на апликација "а". За оваа опција за разделување во примерот, го пресметуваме Z на следниот начин:

TCW = AFF(AP1) + AFF(AP2) + AFF(AP4) = 3 + 7 + 3 = 13

BCW = none = 0

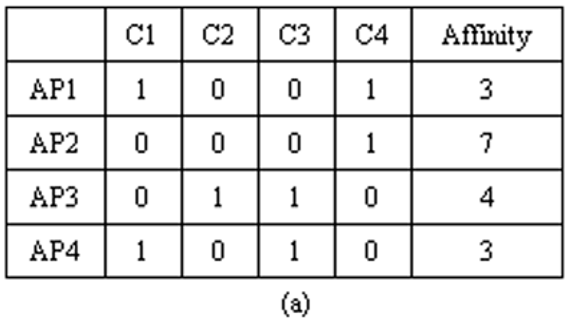
BOCW = AFF(AP3) = 4

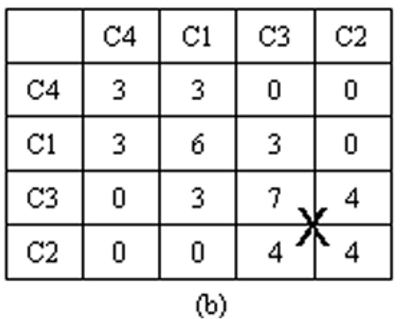
Z = 13\*0 – 42 = –16

Во следниот чекор, точката на разделување се движи кон средината на матрицата, оставајќи ги TC и BC со две колони како што е прикажано на Слика 2.24. Сега, AP1 и AP2 остануваат исти, но AP3 користи колони само за BC, а AP4 користи колони кои се од двата агли. За оваа опција за разделување, вредноста на Z се пресметува на следниов начин:

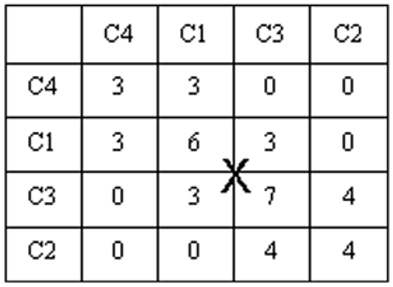
TCW = AFF(AP1) + AFF(AP2) = 3 + 7 = 10

BCW = AFF(AP3) = 4





Слика 33 - Стартни точки за поделбите на групираните афинитетни матрици



Слика 34 - Матрица на втора точка на поделба

BOCW = AFF(AP4) = 3

Z = 4\*10 – 32 = 40 – 9 = 31

Во следниот чекор, точката на разделување се преместува во горниот лев агол како што е прикажано на Слика 2.25. Сега, AP2 сè уште ги користи само колоните од горниот агол, но AP3 и AP4 користат колони само од долниот агол, а AP1 користи колони од двата агли. За оваа опција за разделување имаме следново:

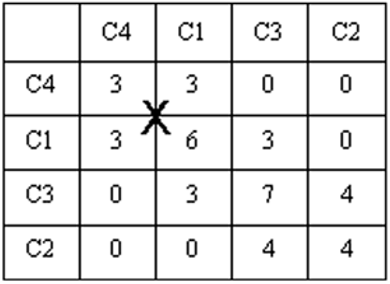
TCW = AFF(AP2) = 7

BCW = AFF(AP3) + AFF(AP4) = 4 + 3 = 7

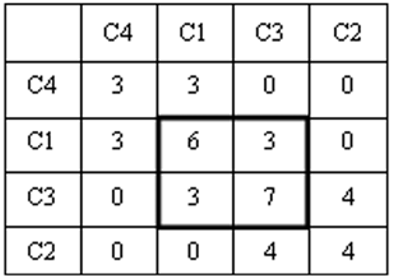
BOCW = AFF(AP1) = 3

Z = 7\*7 – 32 = 49 – 9 = 40

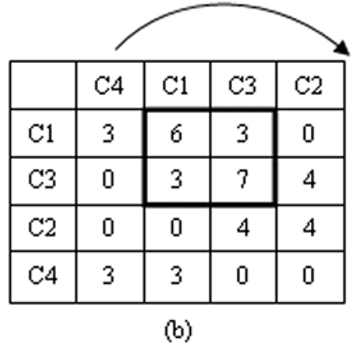
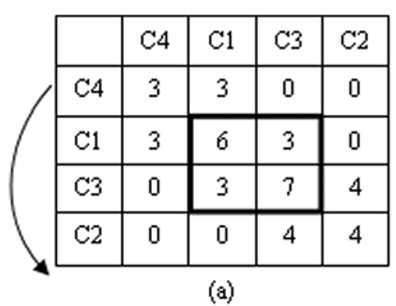
До сега, треба да биде очигледно дека овој пристап создава непреклопување партиции со движење по матрица дијагонала. Предложениот алгоритам има недостаток да не може да издвои вграден или внатрешен блок од колони како партиција. За да може да го стори тоа, пристапот треба да го смести {C1, C3} во една партиција и {C4, C2} во друга партиција како што е прикажано на Слика 36. Можеме да го надминеме овој недостаток со додавање на операција SHIFT. Кога операцијата SHIFT се користи, тој го поместува највисокиот ред на матрицата до дното, а потоа ја поместува левата колона на матрицата до крајната десна страна. Слика 37 ја отсликува оригиналната матрица, заедно со стрелка која покажува каде највисокиот ред треба да се премести како дел од SHIFT. Слика 38 ја покажува матрицата по\*

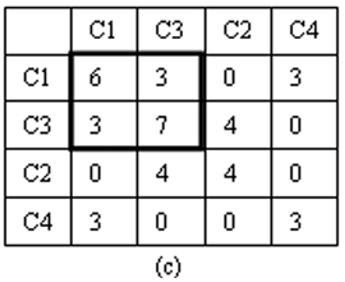


Слика 35 - Матрица на третата точка на поделба



Слика 36 - матрица за внатрешен блок на колони





Слика 37 - Матрица со конечната поделба после SHIFT операцијата

вртењето на највисокиот ред, со друга стрелка која покажува каде најлеви колони треба да се преместат како дел од истата операција SHIFT. Слика 38 ја прикажува матрицата откако десната колона е ротирана. Процесот SHIFT се повторува вкупно n пати, така што секој дијагонален блок добива можност да биде доведен во горниот лев агол во матрицата. За нашиот пример, ние само треба да го искористиме SHIFT еднаш откако сите други комбинации веќе биле покриени. По SHIFT, AP4 користи TC-само колони, AP2 користи BC-само колони, а AP1 и AP3 користат колумни од двете агли. За оваа опција за разделување имаме следново:

TCW = AFF(AP4) = 3

BCW = AFF(AP2) = 7

BOCW = AFF(AP1) + AFF(AP3) = 3 + 4 = 7

Z = 3\*7 – 72 = 21 – 49 = –28

Меѓу сите позитивни Z вредности, максималната вредност е 40. Оваа вредност одговара на опцијата за создавање на две партиции "(C4)" и "(C1, C2, C3)." Како што беше споменато претходно, треба да се вклучи примарното копче на табелата Т во секој вертикален фрагмент. Како резултат на тоа, два вертикални фрагменти ќе бидат дефинирани како “VF1(C, C4)” and “VF2(C, C1, C2, C3).”

# **Вертикална фрагментација правила за корекција**

Бидејќи оригиналната табела не е физички складирана во DDBE, оригиналната табела мора да биде реконструирана од вертикалните фрагменти користејќи комбинација на неколку SQL извештаи (вклучувајќи го и овој случај). Како резултат на тоа, при фрагментирање на табелата вертикално треба да се исполнат следниве барања:

* Комплетност - секој податочен елемент од табелата е барем еден од вертикалните фрагменти.
* Реконструктивност - оригиналната маса може да се реконструира од вертикалните фрагменти.
* Заедничкиот основен клуч – реконструкцијата бара примарниот клуч на табелата да биде копиран во сите вертикални фрагменти.

# **Упатства за генерирање хоризонтална фрагментација.**

Како што беше објаснето претходно, со примена на хоризонтална фрагментација на табелата се создава множество фрагменти кои содржат несогласни редови на табелата (хоризонтална несоодветност). Хоризонталната фрагментација е корисна бидејќи може да ги групира редовите на табелата што ги задоволуваат предикатите на често извршуваните пребарувања. Како такви, сите редови на даден фрагмент треба да бидат во резултатниот сет на барање што често се одвива. За да го искористат ова, дизајнерот на систем на дистрибуирани бази на податоци треба да го зачува секој таков хоризонтален фрагмент на местото каде што се извршуваат овие типови на пребарувања. Ова го покренува прашањето: "Како да одлучиме која состојба или условите да ги користиме кога хоризонтално фрагментираме табела?" Одговорот е формализиран во [Ozsu99] и во [Bobak96]. Ние ќе ја резимираме нивната дискусија во овој дел и ќе ја примениме нивната формулација на пример со цел да го објасниме општиот пристап. За фрагментирање на табелата хоризонтално, ние користиме еден или повеќе предикати (услови). Постојат различни видови на предикати за да се изберат кога ќе се фрагментира табелата. На пример, можеме да користиме едноставни предикати, како што е "Sal> 100000" или "DNO = 1." Во принцип, едноставниот предикат, P, го следи формат "Колона-Име компаративна-оператор вредност." Компаративниот оператор е еден на операторите во собата {=, <,>,> =, <=, <>}. Ние исто така можеме да го искористиме предикатот М, кој е дефиниран како конјугитивна нормална форма на едноставни предикати. На пример, предикатите {Плата> 30.000 Локација = "ЛА"} и {Плата> 30.000 Локација <> "LA"} се минијатурни предикати. Тука операторот се чита како "AND".

Множеството на сите едноставни предикати што ги користат сите апликации што ја бараат зададената табела се прикажани како "Pr = {p1, p2,. . . , pn}. "Множеството на сите minterm предикати што ги користат сите апликации што ја бараат табелата се прикажани како" M = {m1, m2,. . . , mk}. "Применувањето на митермичките предикати М на табелата генерира к minterm хоризонтални фрагменти означени

од множеството "F = {F1, F2,. . . , Fk}. "За овој дизајн на фрагментација, сите редови во Fi, за" i = 1..k, "задоволуваат mi.

## **Минималност и комплетност на хоризонталната фрагментација.**

Треба да биде очигледно дека колку повеќе фрагменти постојат во системот, толку повеќе време системот треба да го потроши за реконструкција на табелата. Како резултат на тоа, важно е да имате минимална вредност

сет на хоризонтални фрагменти. За да генерирате минимален сет на фрагменти, се применуваат следните правила.

*Правило 1.* Редовите на табелата (или фрагментот) треба да се поделат на најмалку два хоризонтални фрагменти, ако на редовите се пристапува различно од најмалку една апликација. Кога последователната примена на Правило 1 повеќе не е потребна, дизајнерот создал минимален и комплетен сет на хоризонтални фрагменти. Затоа, комплетноста е дефинирана како што следува.

*Правило 2.* Збир на едноставни предикати, Pr, за табелата е завршена ако и само ако, за било кој два реда во рамките на секој minterm фрагмент дефиниран на Pr, редовите имаат иста веројатност да се пристапи со било која апликација. Пример 2.7 Да претпоставиме дека апликацијата "AP1" ја испитува табелата "EMP" (види слика 2.3), во потрага по оние кои работат во Лос Анџелес (Лос Анџелес). Поставувањето "Pr = {p1: Loc =" LA "} ги покажува сите потребни едноставни предикати што ги користи AP1. Затоа, множеството "M = {m1: Loc =" LA ", м2: Loc <>" LA "} е минимален и комплетен збир на minterm предикати за AP1. М фрагменти ЕМП во следните два фрагмента:

Fragment F1: Create table LA\_EMPS as

Select \* from EMP

Where Loc = "LA";

Fragment F2: Create table NON\_LA\_EMPS as

Select \* from EMP

Where Loc <> "LA";

Ако AP1 исто така требало (покрај проверката на вредноста на Loc) да исклучи било кој вработен чија плата е помала или еднаква на 30000, тогаш збир на предикати повеќе нема да биде минимален или целосен. Оваа дополнителна проверка би значела дека на редовите во Формула 1 ќе има пристап до AP1 поинаку, во зависност од платата на секој вработен во F1. Применувајќи го правилото за минималноста споменато погоре, ние би требало дополнително да ја фрагментираме EMP табелата, ако е така. Новите едноставни предикати за AP1 би барале промена на Pr и M на следново:

Pr = {p1: Loc = "LA",

p2: salary > 30000}

M = {m1: Loc = "LA" Sal > 30000,

m2: Loc = "LA" Sal <= 30000,

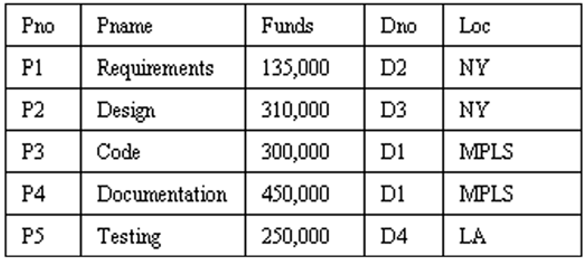
m3: Loc <>"LA" Sal > 30000,

m4: Loc <>"LA" Sal <= 30000}

**Набљудување 1**: Бидејќи во Pr постојат два едноставни предикати, и бидејќи за секој предикат треба да го земеме предвид предикатот и неговиот негативен, М сега ќе има четири минијатурни предикати во него. Во принцип, ако има N едноставни предикати во Pr, M ќе има 2N minterm предикати. Како што е објаснето подоцна во овој дел, не сите минерални предикати се релевантни - ирелевантните предикати мора да се отстранат од сетот. Затоа, вистинскиот број на хоризонтални фрагменти обично е помал од 2N.

**Набљудување 2**: При формирање на минијатурни предикати, се користи само конјуктивална нормална форма ("") и не се употребува дисјугентивна нормална форма ("ИЛИ"). Тоа е затоа што дисјугентивните нормални форми создаваат поостри фрагменти од фрагменти генерирани од конјуктивата нормална форма и оттаму не се потребни.

*Пример 8 :* Како друг пример, треба да се разгледа табелата "PROJ (PNO, Pname, Funds, Dno, Loc)", каде колоната PNO е основен клуч како што е прикажано на Слика 38. Исто така, претпоставете две апликации ("AP1" и "AP2") ја пребаруваат PROJ табелата врз основа на следниве множества на едноставни предикати:



Слика 38 - PROJ табела

The AP1’s simple predicates:

p1: Loc = "MPLS"

p2: Loc = "NY"

p3: Loc = "LA"

The AP2’s simple predicates:

p4: Funds <= 300000

Двете апликации имаат комбиниран сет од четири едноставни предикати во Pr, дефинирани како:

Pr = {Loc = "MPLS",

Loc ="NY",

Loc ="LA",

Funds <= 300000}

Со оглед на четирите едноставни предикати во Pr, M ќе има "24 = 16" различни минијатурни предикати. Следново ги опишува сите овие предикати:

m1 = {Loc = "MPLS" ^ Loc = "NY" ^ Loc = "LA" ^ Funds <= 300000}

m2 = {Loc = "MPLS" ^ Loc = "NY" ^ Loc = "LA" ^ Funds > 300000}

m3 = {Loc = "MPLS" ^ Loc = "NY" ^ Loc <> "LA" ^ Funds <= 300000}

m4 = {Loc = "MPLS" ^ Loc = "NY" ^ Loc <> "LA" ^ Funds > 300000}

m5 = {Loc = "MPLS" ^ Loc <> "NY" ^ Loc = "LA" ^ Funds <= 300000}

m6 = {Loc = "MPLS" ^ Loc <> "NY" ^ Loc = "LA" ^ Funds > 300000}

m7 = {Loc = "MPLS" ^ Loc <> "NY" ^ Loc <> "LA" ^ Funds <= 300000}

m8 = {Loc = "MPLS" ^ Loc <> "NY" ^ Loc <> "LA" ^ Funds > 300000}

m9 = {Loc <> "MPLS" ^ Loc = "NY" ^ Loc = "LA" ^ Funds <= 300000}

m10= {Loc <> "MPLS" ^ Loc = "NY" ^ Loc = "LA" ^ Funds > 300000}

m11= {Loc <> "MPLS" ^ Loc = "NY" ^ Loc <> "LA" ^ Funds <= 300000}

m12= {Loc <> "MPLS" ^ Loc = "NY" ^ Loc <> "LA" ^ Funds > 300000}

m13= {Loc <> "MPLS" ^ Loc <> "NY" ^ Loc = "LA" ^ Funds <= 300000}

m14= {Loc <> "MPLS" ^ Loc <> "NY" ^ Loc = "LA" ^ Funds > 300000}

m15= {Loc <> "MPLS" ^ Loc <> "NY" ^ Loc <> "LA" ^ Funds <= 300000}

m16= {Loc <> "MPLS" ^ Loc <> "NY" ^ Loc <> "LA" ^ Funds > 300000}

**Набљудување 3**: Бидејќи Loc може да има да има една вредност во исто време, можеме да ги поедноставиме предикатите со следното:

m7 = {Loc = "MPLS" ^ Funds <= 300000}

m8 = {Loc = "MPLS" ^ Funds > 300000}

m11= {Loc = "NY" ^ Funds <= 300000}

m12= {Loc = "NY" ^ Funds > 300000}

m13= {Loc = "LA" ^ Funds <= 300000}

m14= {Loc = "LA" ^ Funds > 300000}

Применувајќи ги овие предикати на PROJ, можеме да генерираме фрагменти прикажани на Слика 39. И покрај тоа што постојат шест предитермини, само пет од нив произведуваат корисни резултати од ПРОЈ. За моменталната состојба на табелата, само петте прикажани фрагменти всушност содржат редови, шестиот фрагмент не е. Оваа фрагментација е минимална, бидејќи на редовите во рамките на секој фрагмент се пристапува истото од двете апликации. Оваа фрагментација е исто така комплетна бидејќи секој два реда во рамките на секој фрагмент имаат иста веројатност за пристап за AP1 и AP2.

# **Правила за точност на хоризонталната фрагментација**

Секогаш кога се користи фрагментација (вертикална, хоризонтална или хибридна), оригиналната табела не е физички складирана. Затоа, оригиналната табела мора да се реконструира од фрагменти користејќи комбинација на SQL извештаи. Вертикално фрагментираните табели се реконструираат со користење на операциите за приклучување, хоризонтално фрагментираните табели се реконструираат со користење операции на унии, а табелите фрагментирани со хибридна фрагментација се реконструираат со користење на комбинација од операции на соединување и приклучување. Секоја фрагментација мора да ги задоволи следниве правила како што е дефинирано од Ozsu [¨ Ozsu99]: ¨

**Правило 1**: Комплетноста. Разградување на R во R1, R2,. . . , Rn е завршена ако

и само ако, секоја податотека во R може да се најде и во некои R1.

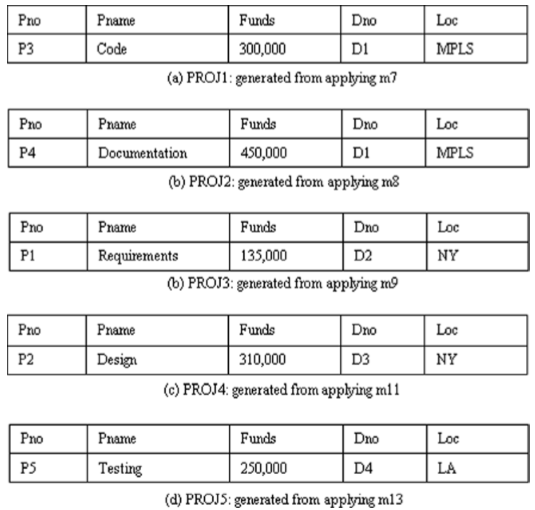
**Правило 2**: Реконструкција. Ако R е распаднат на R1, R2,. . . , Rn, тогаш таму

треба да постои некој релациски оператор , така што "R = 1≤i≤n Ri."

**Правило 3**: Несогласување. Ако R е распаднат на R1, R2,. . . , Rn, и di е торка

во Rj, тогаш di не треба да биде во кој било друг фрагмент, како што е Rk, каде k = j.

Страна 66 ќе видиш не може да се копираат тие...



Слика 39 - Фрагментација на PROJ табели базирана на минтерм предикати

Правилото 1 наведува дека за време на фрагментацијата нема изгубено ниту еден од податоците во оригиналната табела. Секоја податотека која постои во оригиналната табела е во барем еден од фрагментите. Ние го употребуваме терминот "барем" бидејќи вертикалната фрагментација секогаш бара вклучување на колоната на примарен клуч во сите вертикални фрагменти.

Правилото 2 е потребно за реконструкција. Ова правило е потребно затоа што, по фрагментацијата, оригиналната табела повеќе не е зачувана во системот. Локалните DBMS сервери ќе содржат фрагменти како дел од нивната локална концептуална шема. На глобално ниво, корисниците на системот на дистрибуирани бази на податоци не се свесни за фрагментацијата што била применета. За нив, оригиналната табела постои како целина. Тие ќе ја испитаат оригиналната табела, а не фрагментите. Затоа, системот мора да биде во можност да ја реконструира оригиналната табела од своите фрагменти на летот.

Правилото 3 се однесува на хоризонтални фрагменти. Секој хоризонтален фрагмент е генериран со примена на предикат на минтер на оригиналната табела. Затоа, секој хоризонтален фрагмент ги содржи само редовите кои го задоволуваат соодветниот предикат. Бидејќи ова е точно за сите хоризонтални фрагменти, редовите не се споделуваат низ хоризонталните фрагменти.

# **Репликација**

За време на процесот на дизајнирање на базата на податоци, дизајнерот може да одлучи да копира дел од фрагментите или табелите за да обезбеди подобра достапност и сигурност. Треба да биде очигледно дека колку повеќе копии на табелата / фрагментот се создаваат, толку полесно е да се побара таа табела / фрагмент. Од друга страна, колку повеќе копии постојат, толку покомплицирано е да се ажурираат сите копии. Затоа дизајнерот треба да ја знае фреквенцијата со која се бара табела / фрагмент наспроти фреквенцијата со која се менува - преку вметнувања, надградби или бришење. Како правило, ако се бара почесто отколку што е модифицирано, тогаш се препорачува репликација. Откако ќе складираме повеќе од една копија од табела / фрагмент во системот на дистрибуирани бази на податоци, ја зголемуваме веројатноста за копирање на локално ниво за пребарување. Имајќи повеќе од една копија на фрагмент во системот ја зголемува и еластичноста на системот. Тоа е затоа што веројатноста на сите копии да не успеат во исто време е многу мала. Со други зборови, сеуште можеме да пристапиме до една од копиите, дури и ако некои од копиите не успеале: тоа е, се разбира, ако сите копии од фрагментот ги покажуваат истите вредности. Затоа, оваа корист доаѓа со дополнителни трошоци за чување на сите копии идентични. Оваа цена, која потенцијално може да биде висока, се состои од вкупни трошоци за складирање, трошоци за локална обработка и трошоци за комуникација. Забележете дека копиите треба да бидат идентични само кога копиите се онлајн. Ние ќе разговараме за деталите за тоа како копиите се чуваат во синхронизација како дел од контролата на репликација. Ние исто така ќе разговараме за пресметување на вкупните трошоци на пребарувања / ажурирања како дел од управувањето со трансакции и оптимизација на барањето.

# **Заклучок**

Технологијата на дистрибуирани бази на податоци се очекува да има значително влијание врз обработката на податоците во претстојните години. Денешното деловно опкружување има зголемена потреба за дистрибуирани бази на податоци и апликации за клиент / сервер, бидејќи желбата за конзистентни, скалабилни, сигурни и достапни информации постојано расте. Дистрибуираната обработка е ефикасен начин за подобрување на сигурноста и перформансите на системот на бази на податоци. Дистрибуција на податоци е збирка на процеси на фрагментација, распределба и репликација. Претходни истражувања обезбедија решение за фрагментација врз основа на емпириски податоци за видот и зачестеноста на барањата доставени до централизиран систем.

Фрагментацијата може да биде хоризонтална, вертикална или хибридна (мешавина на хоризонтална и вертикална).

Вертикалната фрагментација (VF) ги групира колоните на табелата во фрагменти. VF мора да се направи на таков начин што оригиналната табела може да се реконструира од фрагментите.

HF ги групира редовите на табела врз основа на вредностите на една или повеќе колони. Слично на вертикалната фрагментација, хоризонталната фрагментација мора да се направи на таков начин што основната табела може да се реконструира (реконструкција). Бидејќи секој фрагмент содржи подмножество од редовите во табелата, HF може да се користи за да се спроведе сигурност и / или приватност на податоците. Секој хоризонтален фрагмент мора да ги има сите колони од оригиналната база. Хоризонталната фрагментација понатаму може да се класифицира во две класи: примарна хоризонтална фрагментација (PHF) и изведена хоризонтална фрагментација (DHF).

Хибридната фрагментација (HyF) користи комбинација на хоризонтална и вертикална фрагментација за генерирање на фрагментите што ни се потребни. Постојат два пристапа за тоа. Во првиот пристап, ние генерираме група хоризонтални фрагменти, а потоа вертикално фрагментираме еден од повеќе од овие хоризонтални фрагменти. Во вториот пристап, ние генерираме група вертикални фрагменти, а потоа хоризонтално фрагментираме еден или повеќе од овие вертикални фрагменти. Во секој случај, финалните фрагменти произведени се исти. Овој пристап за фрагментација обезбедува најмногу флексибилност за дизајнерите, но во исто време тоа е најскапиот пристап во однос на реконструкцијата на оригиналната табела.

# **References**

|  |  |
| --- | --- |
| [1] | [Online]. Available: https://www.tutorialspoint.com/distributed\_dbms/distributed\_dbms\_design\_strategies.htm. |
| [2] | F. S. H. SAEED K. RAHIMI, DISTRIBUTED DATABASE MANAGEMENT SYSTEMS: A practical aproach, John Wiley, 2010. |
| [3] | [Online]. Available: https://www.c-sharpcorner.com/article/understanding-fragmentation-in-distributed-databases/. |
| [4] | P. V. M. Tamer Özsu, Principles of Distributed Database Systems, Springer, 2011. |