**NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA**

**Department of Information Technology**



Advanced Database Systems

Assignment 1

ADBS - 1 CAR RENTAL SYSTEM

Mentor Name: Ananthanarayana V.S Submitted by:

Mr. Krish Kumar(222IT015)

Mr. Nilesh Pingale(222IT027)

Contents

[Problem Description: 2](#_Toc121610599)

[Data Sources : 2](#_Toc121610600)

[Actors : 2](#_Toc121610601)

[Queries : 2](#_Toc121610602)

[Constraints : 3](#_Toc121610603)

[Entity-Relationship Diagram: 3](#_Toc121610604)

[Conceptual Schema: 3](#_Toc121610605)

[Normalization: 4](#_Toc121610606)

[Global Schema: 5](#_Toc121610607)

[Fragmentation: 8](#_Toc121610608)

[Horizontal Fragmentation: 11](#_Toc121610609)

[Vertical Fragmentation: 11](#_Toc121610610)

[Data Allocation & Replication: 33](#_Toc121610611)

[Redundant All Beneficial Site method: 35](#_Toc121610612)

[Physical Design: 38](#_Toc121610613)

[Work Area Space and System specification: 42](#_Toc121610614)

# Problem Description:

Online Car Booking management System is developed to manage all cab hiring work online. It is useful for car booking agencies that are specialized in Hiring cabs to customers. Using this system many car-booking agencies are moving ahead to become a pioneer in the vehicle rental industry by completely focusing on customers. Using this system it is very easy for customers to book a car online.

The main problem that we are trying to solve is that some small groups and companies are involved in the entire cab hiring process which are doing some specific tasks.

* Let’s say there is one company ‘A’ which handles all the cab-related services like if anyone wants to give their cabs on rent then they go to company ‘A’. So to maintain this, company ‘A’ keeps the details of all the cabs under a single entity ‘CAB’ like its id, model, year of manufacture, type of car, registration number, fare etc. They also keep records about owners of the cabs. That can be a single person or a company. For this ‘OWNER’ entity is made having the details like id, name, type of owner, number of cabs owned etc are also stored.
* Let’s say there is one company ‘B’ which handles all the drivers related work. It hires the drivers and whenever someone wants a driver then they can contact company ‘B’. This company manages everything related to drivers only. If someone wants to work as a driver then they will approach ‘B’ for that. They keep the data about the drivers in the ‘DRIVER’ like his/her id, name, contact no., age, gender, licence no., their ratings given by customers etc.
* Let’s say there is one company ‘C’ which handles all the payments related work only. It provides payment services for various customers. They provide interface and digital solutions for all kinds of payments. For this the ‘PAYMENT’ entity has details like reference id, date, time, mode of payment, amount, discount given etc are stored in their own database.

So if a person wants to hire a cab, he’ll have to book a cab from ‘A’, hire one driver from ‘B’ and do the payments for cab and driver with the help of ‘C’. Here, we can see that different companies are solving small parts of the problem. All the data which a company maintains is also located at different locations using different format, structure and schema. So we need to use all of them and integrate them to complete one single task ( i.e cab booking ) efficiently. We have to make all these separately located groups communicate with each other because of the data and control flow between multiple locations. So that the customers feel like using a single standalone platform.

# Data Sources :

According to the problem statement, there can be four different data sources.

1. User Information:
   1. It has all the information about customers. It also has the details of the trips completed by the users. The ‘USER’ entity has attributes like id, name, contact no., gender, age etc. The ‘TRIP’ entity has details like id, source, destination, start time, end time, date, distance etc.
2. Driver Information :
   1. It has the driver's details which is handled by company ‘B’.
3. Cab Information:
   1. This location has the data about the cabs and their owners managed by company

‘A’.

1. Payment Information :
   1. All the payments related data is maintained at a specific location by company ‘C’.

# Actors :

* Users/Customers
* Drivers
* Owners

# Queries :

● Simple Queries:

1. Who owns more than 5 cabs?
2. List out all available taxies of type ‘SUV’.
3. How many ‘Male’ drivers are having the rating greater than 3.
4. What is the source and destination of the longest trip?

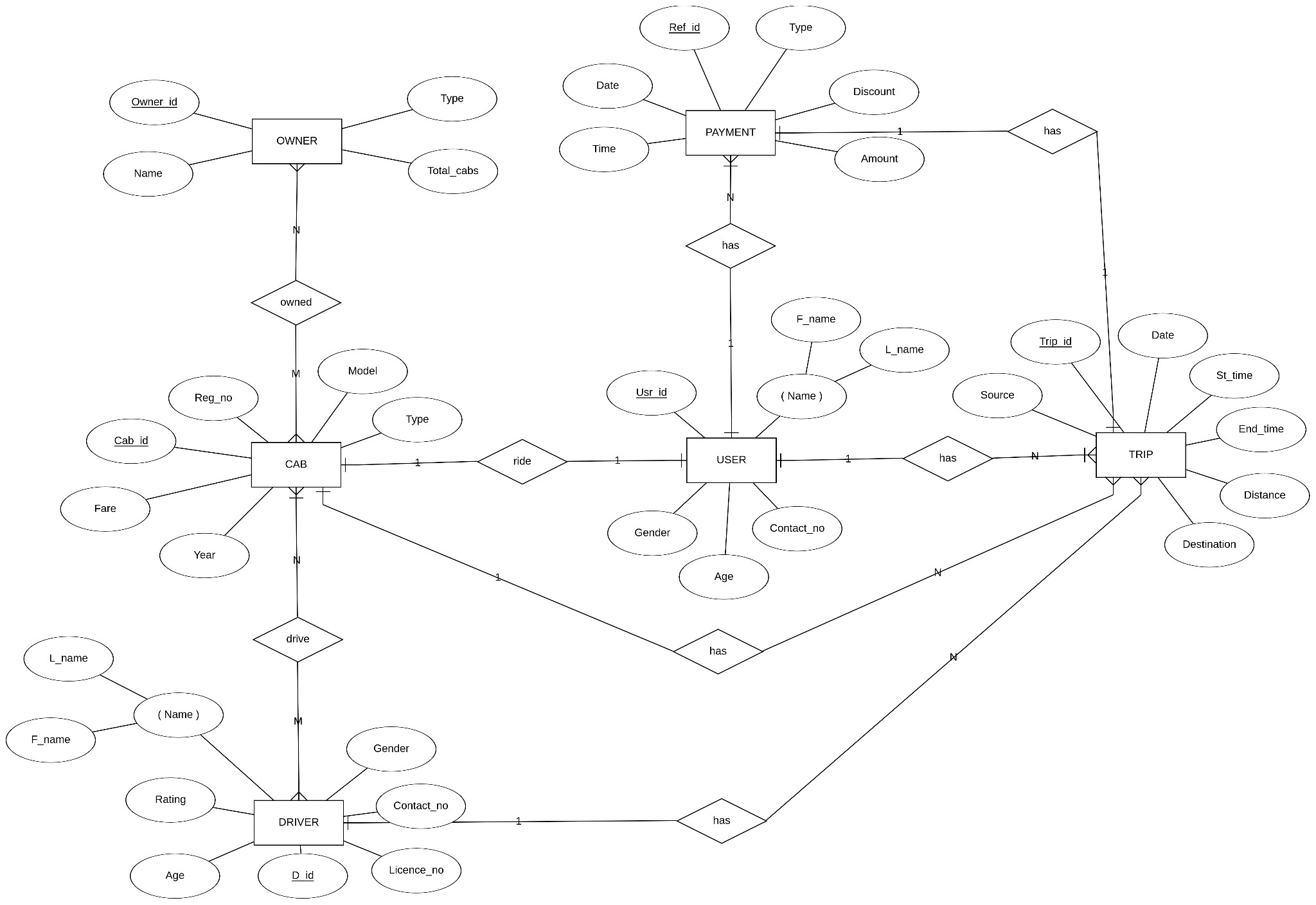
● Complex Queries:

1. How many female users had female drivers?
2. During which trip the most discount was given?
3. Which type of cab is most preferred by customers?

# Constraints :

1. Age of the customer should be greater than 15 years.
2. Age of the driver should be between 18 - 50 years.
3. Discount can not be more than the bill amount.
4. Distance must be more than 1 kilometer.

# Entity-Relationship Diagram:



# Conceptual Schema:

1. User

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| User-id | Name | Contact\_no | Age | Gender |

1. Payment

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Ref-id | User-id | Trip\_id | Date | Time | Type | Amount | Discount |

1. Driver

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| D-id | Name | Contact\_no | Age | Gender | Rating | Licence\_no |

1. Cab

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Cab-id | Owner\_id | Reg\_no | Model | Type | Year | Fare |

1. Owner

|  |  |  |  |
| --- | --- | --- | --- |
| Owner-id | Name | Type | Total\_cabs |

1. Trip

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Trip -id | Cab  \_id | User \_id | Source | Destination | Distance | Start\_ time | End\_ time | Date |

1. Cab-Driver

|  |  |
| --- | --- |
| Cab-id | D-id |

# Normalization:

Why do we need normalization in a database? Normalization removes any type of error, redundancy or anomaly that might exist in the tables of our database. In addition, normalization makes the database more logical, reducing their size and simplifying the schema of the tables to make things easier to operate upon. Objectives of Normalization:

* To correct duplicate data and anomalies.
* To avoid creating and updating any unwanted data dependencies.
* To optimize storage space.
* To facilitate the access and interpretation of data to users and applications thatmake use of databases

**1st Normal Form:**

If a relation contains multi-valued or composite attributes, it violates 1st normal form. A relation is in 1st normal form if every attribute in that relation is a single valued attribute. In our schema design orders table violates 1st normal form. Conditions of 1st normal form:

▪ No multi-valued attribute.

▪ No composite attributes.

▪ Identify a primary key

There is no such attribute having multiple values and neither of them are composite. So all the tables are already in 1st Normal form.

**2nd Normal Form:**

A relation in 2nd normal form if it follows the below conditions.

▪ It is already in the second normal form.

▪ Every non-prime attribute is fully functionally dependent on the primary key.

Basically, we want to eliminate all the partial functional dependencies in our database. All our relations in the database are already in 2nd normal form because every relation has a single attribute primary key, due to which we can say that all non-prime attributes will be functionally dependent on the primary key. Hence, all our relations are already in 2nd Normal Form.

**3rd Normal Form:**

For a relation to be in 3rd normal form it should satisfy the following conditions ▪ It should already be in 2nd Normal Form.

▪ The relation shouldn’t contain any transitive dependencies: non-prime attributes transitively depending on the key.

3 rd Normal form should hold the condition, if X->Y then: Either X is a super key or Y is a prime attribute. By using this rule, we can eliminate all transitive functional dependencies.

There are no transitive dependencies in our database so all are in 3rd Normal Form.

# Global Schema:

* User

|  |  |
| --- | --- |
| **Attribute name** | **size** |
| User-id | int(8) |
| Name | char(20) |
| Contact\_no | int(10) |
| Age | int(2) |
| Gender | char(6) |

* Payment

|  |  |
| --- | --- |
| **Attribute name** | **size** |
| Ref-id | int(8) |
| User\_id | int(8) |
| Trip\_id | int(8) |
| Date | date(10) |
| Time | int(4) |
| Type | char(20) |
| Amount | int(8) |
| Discount | int(8) |

* Driver

|  |  |
| --- | --- |
| **Attribute name** | **size** |
| D-id | int(8) |
| Name | char(20) |
| Contact\_no | int(10) |
| Age | int(2) |
| Gender | char(6) |
| Rating | int(2) |
| Licence\_no | int(15) |

* Cab

|  |  |
| --- | --- |
| **Attribute name** | **size** |
| Cab-id | int(8) |
| Reg\_no | int(8) |
| Model | char(20) |
| Type | char(20 |
| Year | int(4) |
| Fare | int(8) |

* Owner

|  |  |
| --- | --- |
| **Attribute name** | **size** |
| Owner-id | int(8) |
| Name | char(20) |
| Type | char(6) |
| Total\_cabs | int(4) |

* Trip

|  |  |
| --- | --- |
| **Attribute name** | **size** |
| Trip-id | int(8) |
| Cab-id | int(8) |
| User-id | int(8) |
| Source | char(20) |
| Destination | char(20) |
| Distance | int(5) |
| Start\_time | time(4) |
| End\_time | time(4) |
| Date | date(10) |

* Cab-Driver

|  |  |
| --- | --- |
| **Attribute name** | **size** |
| Cab-id | int(8) |
| D-id | int(8) |

# Fragmentation:

The main goal of DDBMS is to provide the data to the user involving less overhead and as quickly as possible. This is provided by fragmentation of data. Data Fragmentation provides distribution transparency of the data over the database. Dividing the whole table into smaller chunks and storing them in different Databases in the Distributed Database Management System is called data fragmentation.

Fragmentation of data provides the following advantages:

* Storage will not be exhausted quickly.
* Provides parallel processing.
* Provides Load balancing.
* Improves query response time.
* Provides better local processing.
* Better availability of data.

Decomposed fragments are placed into some other site to facilitate query and optimize other quality of services. These fragments permit a number of transactions concurrently. Taking a copy of a relation and maintaining it in another site is called replication. One can combine fragmentation and replication for better service provision. There are two kinds of fragmentation: horizontal and vertical.

They must satisfy the following properties:

* Completeness: all rows or columns must be present in at least one site.
* Reconstruction: while reconstructing the relation, there should not be any inconsistency or loss of data.
* Dis-jointness: row or column must be present in at most one site, else will lead to inconsistent data.

Fragmentation takes place in a relation based on the query and its frequency. The predicates used in the query servers are an important statistical input for fragments.

Following are the lists of queries depicting the transactions in the ICC Database Management system.

**Query 1** : Who owns more than 5 cabs?

SELECT Owner\_id, Name

FROM Owner

WHERE Total\_cabs > 5 ;

**Query 2** : List out all available taxies of type ‘SUV’.

SELECT Cab\_id, Model

FROM Cab

WHERE Type = ‘SUV’ ;

**Query 3** : How many ‘Male’ drivers are having the rating greater than 3.

SELECT COUNT(\*)

FROM Driver

WHERE Gender = ‘Male’ AND Rating > 3 ;

**Query 4** : What is the source and destination of the longest trip?

SELECT Source, Destination

FROM Trip

WHERE Distance = ( SELECT MAX(Distance) FROM Trip ) ;

**Query 5** :How many female users had female drivers?

SELECT count(\*) as count

FROM ( SELECT U\_id FROM User AS U WHERE U.Gender=’Female’ ) AS A WHERE A.U\_id IN

(SELECT U\_id FROM Trip AS T WHERE T.D\_id IN

(SELECT D\_id FROM Driver AS D WHERE D.Gender=’Female’)) ;

**Query 6** :During which trip the most discount was given?

SELECT Trip\_id, Source, Destination, MAX(Distance)

FROM Payment AS P

INNER JOIN Trip AS T

ON P.Trip\_id = T.Trip\_id ;

**Query 7** : Which type of cab is most preferred by customers?

SELECT C.Type, COUNT(\*) AS Count

FROM CAB AS C

INNER JOIN TRIP AS T

ON C.Cab\_id = T.Cab\_id

GROUP BY C.Type

ORDER BY Count DESC LIMIT 1 ;

# Horizontal Fragmentation:

Horizontal fragmentation partitions the relation along its tuples of the relations. Every fragment will have the same number of attributes. There are two ways of doing it. Primary and derived horizontal fragmentation. But it is usually done using the predicate defined on the queries. For example, consider Cab relation and it’s attribute Type.

* Fragmentation of Cab relation.

**Type** from **Cab** relation:

Cab 1 : All cabs of type ‘SUV’

Cab 2 : All cabs of type ‘Sedan’

* Fragmentation of Driver Relation.

**Rating** from **Driver** relation**.**

Driver 1 = All drivers with rating > 3.

Driver 2 = All drivers with rating <= 3.

In the same way, based on the other splitting attributes we can horizontally fragment all our other relations:. But, we restrict the horizontal fragmentation to the above mentioned tables.

# Vertical Fragmentation:

The vertical fragmentation of a relation R produces subschemas R1, R2, R2,…Rn.

Each of which contains a subset of attributes, and only one fragment has a candidate key. To satisfy reconstruction, we need to use a joining attribute common between the sub schema. There are two methods to perform vertical fragmentation:

* Grouping (bottom up): performed by combining every two attributes at a timeand takes a long time if the number of attributes are over 100 to get desired fragments.
* Splitting (top down): given all attributes together is taken as a fragment and splitthem as many fragments as you want to get. This is much quicker than the first method.

Inputs to the Vertical Fragmentation step are the Frequency Matrix, the Usage Matrix and the Attribute Affinity Matrix.

* Frequency matrix specifies the frequency measure of each query from each site.
* Usage Matrix specifies the attributes of a relation that a query access.
* Attribute Affinity Matrix specifies the affinity measure of each pair of attributes
* **Frequency Matrix**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **S1** | **S2** | **S3** | **S4** | **Total Query** |
| **Q1** | 0 | 10 | 20 | 0 | 30 |
| **Q2** | 30 | 0 | 10 | 0 | 40 |
| **Q3** | 20 | 25 | 20 | 0 | 65 |
| **Q4** | 15 | 10 | 0 | 5 | 30 |
| **Q5** | 0 | 20 | 15 | 5 | 40 |
| **Q6** | 20 | 0 | 0 | 30 | 50 |
| **Q7** | 30 | 20 | 25 | 0 | 75 |

* **Attribute Usage Matrix:**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Trip-id  A1 | Cab\_ id  A2 | User  \_id  A3 | Source  A4 | Destination  A5 | Distance  A6 | Start\_ time  A7 | End\_ time  A8 | Date  A9 |
| **Q1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Q2** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Q3** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Q4** | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| **Q5** | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Q6** | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| **Q7** | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

* **Attribute Affinity Matrix:**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Trip-id  A1 | Cab\_ id  A2 | User\_ id  A3 | Source  A4 | Destination  A5 | Distance  A6 | Start\_ time  A7 | End\_ time  A8 | Date  A9 |
| **A1** | 50 | 0 | 0 | 50 | 50 | 0 | 0 | 0 | 0 |
| **A2** | 0 | 115 | 40 | 0 | 0 | 0 | 0 | 0 | 0 |
| **A3** | 0 | 40 | 40 | 0 | 0 | 0 | 0 | 0 | 0 |
| **A4** | 50 | 23 | 0 | 80 | 80 | 30 | 0 | 0 | 0 |
| **A5** | 50 | 0 | 0 | 80 | 80 | 30 | 0 | 0 | 0 |
| **A6** | 0 | 0 | 0 | 30 | 30 | 30 | 0 | 0 | 0 |
| **A7** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **A8** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **A9** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Ordering of the attributes:**

* + Attribute A3:

CONT(031): 2[Bond(03)+Bond(31)-Bond(01)] : 0

CONT(132): 2[Bond(13)+Bond(32)-Bond(12)] : 12400

CONT(234): 2[Bond(23)+Bond(34)-Bond(24)] : 12400 The value of CONT(132) is larger. So the order will be T(132).

* + Attribute A4:

CONT(041): 2[Bond(04)+Bond(41)-Bond(01)] : 16000

CONT(142): 2[Bond(14)+Bond(42)-Bond(12)] : 16000

CONT(243): 2[Bond(24)+Bond(43)-Bond(23)] : -12400

CONT(345): 2[Bond(34)+Bond(45)-Bond(35)] : 32000

The value of CONT(345) is larger, so the order of this attribute need not be changed, the order is T(1324).

* + Attribute A5:

CONT(051)=2[Bond(05)+Bond(51)-Bond(01)] : 16000

CONT(152)=2[Bond(15)+Bond(52)-Bond(12)] : 16000

CONT(253)=2[Bond(25)+Bond(53)-Bond(23)] : -12400

CONT(354)=2[Bond(35)+Bond(54)-Bond(34)] : 32400

CONT(456)=2[Bond(45)+Bond(56)-Bond(46)] : 32400

The value of CONT(354) is larger. So the order will be T(13254) ● Attribute A6:

CONT(061)=2[Bond(06)+Bond(61)-Bond(01)] : 0

CONT(162)=2[Bond(16)+Bond(62)-Bond(12)] : 0

CONT(263)=2[Bond(26)+Bond(63)-Bond(23)] : -12400

CONT(364)=2[Bond(36)+Bond(64)-Bond(34)] : 5700

CONT(465)=2[Bond(46)+Bond(65)-Bond(45)] : -9600

CONT(567)=2[Bond(56)+Bond(67)-Bond(57)] : 11400 The value of CONT(567) is larger. So the order will be same T(132546)

* Attribute A7:

CONT(071)=2[Bond(07)+Bond(71)-Bond(01)] : 0

CONT(172)=2[Bond(17)+Bond(72)-Bond(12)] : 0

CONT(273)=2[Bond(27)+Bond(73)-Bond(23)] : 0

CONT(374)=2[Bond(37)+Bond(74)-Bond(34)] : 0

CONT(475)=2[Bond(47)+Bond(75)-Bond(45)] : 0 CONT(576)=2[Bond(57)+Bond(76)-Bond(56)] : 0

CONT(678)=2[Bond(67)+Bond(78)-Bond(58)] : 0

* Attribute A8:

All the values are zero, same as the previous attribute.

* Attribute A9:

All the values are zero, same as the previous attribute.

So, overall the final order of the attributes in the relation is T(132546789).

Reordering the attributes based on the above values will give us:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **A1** | **A3** | **A2** | **A5** | **A4** | **A6** | **A7** | **A8** | **A9** |
| **A1** | 50 | 0 | 0 | 50 | 50 | 0 | 0 | 0 | 0 |
| **A3** | 0 | 40 | 115 | 0 | 0 | 0 | 0 | 0 | 0 |
| **A2** | 0 | 40 | 40 | 0 | 0 | 0 | 0 | 0 | 0 |
| **A5** | 50 | 23 | 0 | 80 | 80 | 30 | 0 | 0 | 0 |
| **A4** | 50 | 0 | 0 | 80 | 80 | 30 | 0 | 0 | 0 |
| **A6** | 0 | 0 | 0 | 30 | 30 | 30 | 0 | 0 | 0 |
| **A7** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **A8** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **A9** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Partition Algorithm:**

|  |  |  |  |
| --- | --- | --- | --- |
| **1.** | **TA= {A1}** | **BA= {A3, A2, A5, A4, A6, A7, A8, A9}** | |
|  | TQ= { } | CTQ=0 | |
|  | BQ= { Q4, Q5, Q7 } | CBQ=145 | |
|  | OQ= { Q6 }  Z = -2500 | COQ=50 | |
| **2.** | **TA= {A1, A3}** | **BA= {A2, A5, A4, A6, A7, A8, A9}** | |
|  | TQ= { } | CTQ=0 | |
|  | BQ= { Q4, Q7 } | CBQ=105 | |
|  | OQ= { Q5, Q6 }  Z = -8100 | COQ=90 | |
| **3.** | **TA= {A1, A3, A2}** | **BA= {A5, A4, A6, A7, A8, A9}** | |
|  | TQ= { Q5, Q7 } | CTQ=115 | |
|  | BQ= { Q4 } | CBQ=30 | |
|  | OQ= { Q6 }  Z = 950 | COQ=50 | |
| **4.** | **TA= {A1, A3, A2, A5}** | **BA= {A4, A6, A7, A8, A9}** | |
|  | TQ= { Q5, Q7 } | CTQ=115 | |
|  | BQ= { } | CBQ=0 | |
|  | OQ= { Q4, Q6 }  Z = -6400 | COQ=80 | |
| **5.** | **TA= {A1, A3, A2, A5, A4}** | **BA= {A6, A7, A8, A9}** | |
|  | TQ= { Q5, Q7, Q6 } | CTQ=165 | |
|  | BQ= { } | CBQ=0 | |
|  | OQ= { Q4 }  Z = -900 | COQ=30 | |
| **6.** | **TA= {A1, A3, A2, A5, A4,A6}** | **BA= {A7, A8, A9}** | |
|  | TQ= { Q4, Q5, Q7, Q6 } | CTQ=195 | |
|  | BQ= { } | CBQ=0 | |
|  | OQ= { }  Z = 0 | COQ=0 | |
| **7.** | **TA= {A1, A3, A2, A5, A4, A6, A7}** | **BA= {A8, A9}** | |
|  | TQ= { Q4, Q5, Q7, Q6 } | CTQ=195 | |
| BQ= { } | CBQ=0 |
| OQ= { } | COQ=0 |

Z = 0

**8. TA= {A1, A3, A2, A5, A4,A6, A7, A8} BA= {A9}**

|  |  |
| --- | --- |
| TQ= { Q4, Q5, Q7, Q6 } | CTQ=195 |
| BQ= { } | CBQ=0 |
| OQ= { } | COQ=0 |

Z = 0

Based on the above procedure of vertical fragmentation on the TRIP relation, we will vertically fragment the TRIP relation: **TRIP1( 132 ) and TRIP2( 546789 ).**

Similarly we will check for other relation table like Cabs , Driver , User , Payment , Owner , Cab-Driver. As we have already horizontally fragment the Cab and Driver.

The frequency matrix will be as its, now we have to calculate the attribute usage matrix, attribute affinity matrix, and cluster affinity matrix

**Attribute usage Matrix for User:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **A1** | **A2** | **A3** | **A4** | **A5** |
| **Q1** | **0** | **0** | **0** | **0** | **0** |
| **Q2** | **0** | **0** | **0** | **0** | **0** |
| **Q3** | **0** | **0** | **0** | **0** | **0** |
| **Q4** | **0** | **0** | **0** | **0** | **0** |
| **Q5** | **1** | **0** | **0** | **0** | **1** |
| **Q6** | **0** | **0** | **0** | **0** | **0** |
| **Q7** | **0** | **0** | **0** | **0** | **0** |

**Attribute affinity matrix for user:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **A1** | **A2** | **A3** | **A4** | **A5** |
| **A1** | **40** | **0** | **0** | **0** | **40** |
| **A2** | **0** | **0** | **0** | **0** | **0** |
| **A3** | **0** | **0** | **0** | **0** | **0** |
| **A4** | **0** | **0** | **0** | **0** | **0** |
| **A5** | **40** | **0** | **0** | **0** | **40** |

cont( A 0 , A 3 , A 1 ) = 2 \* ( 0 + 0.0 - 0 ) = 0.0

cont( A 1 , A 3 , A 2 ) = 2 \* ( 0.0 + 0.0 - 0.0 ) = 0.0

cont( A 2 , A 3 , A 4 ) = 2 \* ( 0.0 + 0.0 - 0.0 ) = 0.0

cont( A 0 , A 4 , A 1 ) = 2 \* ( 0 + 0.0 - 0 ) = 0.0

cont( A 1 , A 4 , A 2 ) = 2 \* ( 0.0 + 0.0 - 0.0 ) = 0.0

cont( A 2 , A 4 , A 3 ) = 2 \* ( 0.0 + 0.0 - 0.0 ) = 0.0

cont( A 3 , A 4 , A 5 ) = 2 \* ( 0.0 + 0.0 - 0.0 ) = 0.0

cont( A 0 , A 5 , A 1 ) = 2 \* ( 0 + 0.0 - 0 ) = 0.0

cont( A 1 , A 5 , A 2 ) = 2 \* ( 0.0 + 0.0 - 0.0 ) = 0.0

cont( A 2 , A 5 , A 3 ) = 2 \* ( 0.0 + 3200.0 - 0.0 ) = 6400.0

cont( A 3 , A 5 , A 4 ) = 2 \* ( 3200.0 + 0.0 - 0.0 ) = 6400.0

cont( A 4 , A 5 , A 6 ) = 2 \* ( 0.0 + 0 - 0 ) = 0.0

so final **cluster affinity matrix** is:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **A4** |  | **A3** | **A5** | **A1** | **A2** |
| **A4** | **40** |  | **0** | **0** | **0** | **40** |
| **A3** | **0** |  | **0** | **0** | **0** | **0** |
| **A5** | **0** |  | **0** | **0** | **0** | **0** |
| **A1** | **0** |  | **0** | **0** | **0** | **0** |
| **A2** | **40** |  | **0** | **0** | **0** | **40** |

**Partition Algorithm**:

Fragments = [A4] [A3 A5 A1 A2]

TQ = []

BQ = [5]

OQ = [1, 2, 3, 4, 6, 7]

z = -84100.0

Fragments = [A4 A3] [A5 A1 A2]

TA = [0, 0, 0, 0, 0, 0, 0]

TB = [0, 0, 0, 0, 1, 0, 0]

TQ = []

BQ = [A5]

OQ = [A1, A2, A3, A4, A6,A7]

z = -84100.0

Fragments = [A4 A3 A5] [A1 A2]

TA = [0, 0, 0, 0, 1, 0, 0]

TB = [0, 0, 0, 0, 1, 0, 0]

TQ = []

BQ = []

OQ = [A1, A2, A3, A4, A5, A6, A7]

z = -108900.0

Fragments = [A4 A3 A5 A1] [A2]

TA = [0, 0, 0, 0, 1, 0, 0]

TB = [0, 0, 0, 0, 0, 0, 0]

TQ = [A5]

BQ = []

OQ = [A1, A2, A3, A4, A6, A7]

z = -84100.0

***so here we see in every case z is negative so fragmentation is not possible***.

**Attribute usage matrix for Payment:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **A1** | **A2** | **A3** | **A4** | **A5** | **A6** | **A7** | **A8** |
| **Q1** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **Q2** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **Q3** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **Q4** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **Q5** | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **Q6** | **0** | **0** | **1** | **0** | **0** | **0** | **0** | **0** |
| **Q7** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |

**Attribute affinity Matrix for Payment:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **A1** | **A2** | **A3** | **A4** | **A5** | **A6** | **A7** | **A8** |
| **A1** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **A2** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **A3** | **0** | **0** | **50** | **0** | **0** | **0** | **0** | **0** |
| **A4** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **A5** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **A6** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **A7** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **A8** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |

To find the cluster affinity matrix we have to use Bond energy algorithm.

cont( A 0 , A 3 , A 1 ) = 2 \* ( 0 + 0.0 - 0 ) = 0.0

cont( A 1 , A 3 , A 2 ) = 2 \* ( 0.0 + 0.0 - 0.0 ) = 0.0

cont( A 2 , A 3 , A 4 ) = 2 \* ( 0.0 + 0.0 - 0.0 ) = 0.0

cont( A 0 , A 4 , A 1 ) = 2 \* ( 0 + 0.0 - 0 ) = 0.0

cont( A 1 , A 4 , A 2 ) = 2 \* ( 0.0 + 0.0 - 0.0 ) = 0.0

cont( A 2 , A 4 , A 3 ) = 2 \* ( 0.0 + 0.0 - 0.0 ) = 0.0

cont( A 3 , A 4 , A 5 ) = 2 \* ( 0.0 + 0.0 - 0.0 ) = 0.0

cont( A 0 , A 5 , A 1 ) = 2 \* ( 0 + 0.0 - 0 ) = 0.0

cont( A 1 , A 5 , A 2 ) = 2 \* ( 0.0 + 0.0 - 0.0 ) = 0.0

cont( A 2 , A 5 , A 3 ) = 2 \* ( 0.0 + 0.0 - 0.0 ) = 0.0

cont( A 3 , A 5 , A 4 ) = 2 \* ( 0.0 + 0.0 - 0.0 ) = 0.0

cont( A 4 , A 5 , A 6 ) = 2 \* ( 0.0 + 0.0 - 0.0 ) = 0.0

cont( A 0 , A 6 , A 1 ) = 2 \* ( 0 + 0.0 - 0 ) = 0.0

cont( A 1 , A 6 , A 2 ) = 2 \* ( 0.0 + 0.0 - 0.0 ) = 0.0

cont( A 2 , A 6 , A 3 ) = 2 \* ( 0.0 + 0.0 - 0.0 ) = 0.0

cont( A 3 , A 6 , A 4 ) = 2 \* ( 0.0 + 0.0 - 0.0 ) = 0.0

cont( A 4 , A 6 , A 5 ) = 2 \* ( 0.0 + 0.0 - 0.0 ) = 0.0

cont( A 5 , A 6 , A 7 ) = 2 \* ( 0.0 + 0.0 - 0.0 ) = 0.0

cont( A 0 , A 7 , A 1 ) = 2 \* ( 0 + 0.0 - 0 ) = 0.0

cont( A 1 , A 7 , A 2 ) = 2 \* ( 0.0 + 0.0 - 0.0 ) = 0.0

cont( A 2 , A 7 , A 3 ) = 2 \* ( 0.0 + 0.0 - 0.0 ) = 0.0

cont( A 3 , A 7 , A 4 ) = 2 \* ( 0.0 + 0.0 - 0.0 ) = 0.0

cont( A 4 , A 7 , A 5 ) = 2 \* ( 0.0 + 0.0 - 0.0 ) = 0.0

cont( A 5 , A 7 , A 6 ) = 2 \* ( 0.0 + 0.0 - 0.0 ) = 0.0

cont( A 6 , A 7 , A 8 ) = 2 \* ( 0.0 + 0.0 - 0.0 ) = 0.0

cont( A 0 , A 8 , A 1 ) = 2 \* ( 0 + 0.0 - 0 ) = 0.0

cont( A 1 , A 8 , A 2 ) = 2 \* ( 0.0 + 0.0 - 0.0 ) = 0.0

cont( A 2 , A 8 , A 3 ) = 2 \* ( 0.0 + 0.0 - 0.0 ) = 0.0

cont( A 3 , A 8 , A 4 ) = 2 \* ( 0.0 + 0.0 - 0.0 ) = 0.0

cont( A 4 , A 8 , A 5 ) = 2 \* ( 0.0 + 0.0 - 0.0 ) = 0.0

cont( A 5 , A 8 , A 6 ) = 2 \* ( 0.0 + 0.0 - 0.0 ) = 0.0

cont( A 6 , A 8 , A 7 ) = 2 \* ( 0.0 + 0.0 - 0.0 ) = 0.0

cont( A 7 , A 8 , A 9 ) = 2 \* ( 0.0 + 0 - 0 ) = 0.0

Cluster Affinity matrix for payment:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **A8** | **A7** | **A6** | **A5** | **A4** | **A3** | **A1** | **A2** |
| **A8** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **A7** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **A6** | **0** | **0** | **50** | **0** | **0** | **0** | **0** | **0** |
| **A5** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **A4** | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **A3** | **0** | **0** | **0** | **0** | **0** | **50** | **0** | **0** |
| **A1** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **A2** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |

**Partition Algorithm:**

Fragments = [A8] [A7 A6 A5 A4 A3 A1 A2]

TA = [0, 0, 0, 0, 0, 0, 0]

TB = [0, 0, 0, 0, 0, 1, 0]

TQ = []

BQ = [A6]

OQ = [A1, A2, A3, A4, A5, A7]

z = -78400.0

Fragments = [A8 A7] [A6 A5 A4 A3 A1 A2]

TA = [0, 0, 0, 0, 0, 0, 0]

TB = [0, 0, 0, 0, 0, 1, 0]

TQ = []

BQ = [A6]

OQ = [A1, A2, A3, A4, A5, A7]

z = -78400.0

Fragments = [A8 A7 A6] [A5 A4 A3 A1 A2]

TA = [0, 0, 0, 0, 0, 0, 0]

TB = [0, 0, 0, 0, 0, 1, 0]

TQ = []

BQ = [A6]

OQ = [A1, A2, A3, A4, A5, A7]

z = -78400.0

Fragments = [A8 A7 A6 A5] [A4 A3 A1 A2]

TA = [0, 0, 0, 0, 0, 0, 0]

TB = [0, 0, 0, 0, 0, 1, 0]

TQ = []

BQ = [A6]

OQ = [A1, A2, A3, A4, A5, A7]

z = -78400.0

Fragments = [A8 A7 A6 A5 A4] [A3 A1 A2]

TA = [0, 0, 0, 0, 0, 0, 0]

TB = [0, 0, 0, 0, 0, 1, 0]

TQ = []

BQ = [A6]

OQ = [A1, A2, A3, A4, A5, A7]

z = -78400.0

Fragments = [A8 A7 A6 A5 A4 A3] [A1 A2]

TA = [0, 0, 0, 0, 0, 1, 0]

TB = [0, 0, 0, 0, 0, 0, 0]

TQ = [A6]

BQ = []

OQ = [A1, A2, A3, A4, A5, A7]

z = -78400.0

Fragments = [A8 A7 A6 A5 A4 A3 A1] [A2]

TA = [0, 0, 0, 0, 0, 1, 0]

TB = [0, 0, 0, 0, 0, 0, 0]

TQ = [A6]

BQ = []

OQ = [A1, A2, A3, A4, A5, A7]

z = -78400.0

***so here we see in every case z is negative so Vertical Fragmentation not possible.***

**Attribute usage matrix for Driver relation**:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **A1** | **A2** | **A3** | **A4** | **A5** | **A6** | **A7** |
| **Q1** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **Q2** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **Q3** | **0** | **0** | **0** | **0** | **1** | **1** | **0** |
| **Q4** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **Q5** | **1** | **0** | **0** | **0** | **0** | **1** | **0** |
| **Q6** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **Q7** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |

**Attribute affinity matrix for driver**:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **A1** | **A2** | **A3** | **A4** | **A5** | **A6** | **A7** |
| **A1** | **40** | **0** | **0** | **0** | **40** | **0** | **0** |
| **A2** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **A3** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **A4** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **A5** | **40** | **0** | **0** | **0** | **105** | **65** | **0** |
| **A6** | **0** | **0** | **0** | **0** | **65** | **65** | **0** |
| **A7** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |

cont( A 0 , A 3 , A 1 ) = 2 \* ( 0 + 0.0 - 0 ) = 0.0

cont( A 1 , A 3 , A 2 ) = 2 \* ( 0.0 + 0.0 - 0.0 ) = 0.0

cont( A 2 , A 3 , A 4 ) = 2 \* ( 0.0 + 0.0 - 0.0 ) = 0.0

cont( A 0 , A 4 , A 1 ) = 2 \* ( 0 + 0.0 - 0 ) = 0.0

cont( A 1 , A 4 , A 2 ) = 2 \* ( 0.0 + 0.0 - 0.0 ) = 0.0

cont( A 2 , A 4 , A 3 ) = 2 \* ( 0.0 + 0.0 - 0.0 ) = 0.0

cont( A 3 , A 4 , A 5 ) = 2 \* ( 0.0 + 0.0 - 0.0 ) = 0.0

cont( A 0 , A 5 , A 1 ) = 2 \* ( 0 + 0.0 - 0 ) = 0.0

cont( A 1 , A 5 , A 2 ) = 2 \* ( 0.0 + 0.0 - 0.0 ) = 0.0

cont( A 2 , A 5 , A 3 ) = 2 \* ( 0.0 + 5800.0 - 0.0 ) = 11600.0

cont( A 3 , A 5 , A 4 ) = 2 \* ( 5800.0 + 0.0 - 0.0 ) = 11600.0

cont( A 4 , A 5 , A 6 ) = 2 \* ( 0.0 + 11050.0 - 0.0 ) = 0.0

cont( A 0 , A 6 , A 1 ) = 2 \* ( 0 + 0.0 - 0 ) = 0.0

cont( A 1 , A 6 , A 2 ) = 2 \* ( 0.0 + 0.0 - 0.0 ) = 0.0

cont( A 2 , A 6 , A 3 ) = 2 \* ( 0.0 + 11050.0 - 0.0 ) = 22100.0

cont( A 3 , A 6 , A 4 ) = 2 \* ( 11050.0 + 2600.0 - 5800.0 ) = 15700.0

cont( A 4 , A 6 , A 5 ) = 2 \* ( 2600.0 + 0.0 - 0.0 ) = 5200.0

cont( A 5 , A 6 , A 7 ) = 2 \* ( 0.0 + 0.0 - 0.0 ) = 0.0

cont( A 0 , A 7 , A 1 ) = 2 \* ( 0 + 0.0 - 0 ) = 0.0

cont( A 1 , A 7 , A 2 ) = 2 \* ( 0.0 + 0.0 - 0.0 ) = 0.0

cont( A 2 , A 7 , A 3 ) = 2 \* ( 0.0 + 0.0 - 0.0 ) = 0.0

cont( A 3 , A 7 , A 4 ) = 2 \* ( 0.0 + 0.0 - 11050.0 ) = -22100.0

cont( A 4 , A 7 , A 5 ) = 2 \* ( 0.0 + 0.0 - 5800.0 ) = -11600.0

cont( A 5 , A 7 , A 6 ) = 2 \* ( 0.0 + 0.0 - 0.0 ) = 0.0

cont( A 6 , A 7 , A 8 ) = 2 \* ( 0.0 + 0 - 0 ) = 0.0

**Cluster Affinity matrix for Driver**:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **A7** | **A4** | **A3** | **A6** | **A5** | **A1** | **A2** |
| **A7** | **40** | **0** | **0** | **0** | **40** | **0** | **0** |
| **A4** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **A3** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **A6** | **0** | **0** | **0** | **65** | **65** | **0** | **0** |
| **A5** | **40** | **0** | **0** | **65** | **105** | **40** | **0** |
| **A1** | **0** | **0** | **0** | **0** | **40** | **40** | **0** |
| **A2** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |

Fragments = [A7] [A4 A3 A6 A5 A1 A2]

TA = [0, 0, 0, 0, 0, 0, 0]

TB = [0, 0, 1, 0, 1, 0, 0]

TQ = []

BQ = [A3, A5]

OQ = [A1, A2, A4, A6, A7]

z = -50625.0

Fragments = [A7 A4] [A3 A6 A5 A1 A2]

TA = [0, 0, 0, 0, 0, 0, 0]

TB = [0, 0, 1, 0, 1, 0, 0]

TQ = []

BQ = [A3, A5]

OQ = [A1, A2, A4, A6, A7]

z = -50625.0

Fragments = [A7 A4 A3] [A6 A5 A1 A2]

TA = [0, 0, 0, 0, 0, 0, 0]

TB = [0, 0, 1, 0, 1, 0, 0]

TQ = []

BQ = [A3, A5]

OQ = [A1, A2, A4, A6, A7]

z = -50625.0

Fragments = [A7 A4 A3 A6] [A5 A1 A2]

TA = [0, 0, 1, 0, 0, 0, 0]

TB = [0, 0, 1, 0, 1, 0, 0]

TQ = []

BQ = [A5]

OQ = [A1, A2, A3, A4, A6, A7]

z = -84100.0

Fragments = [A7 A4 A3 A6 A5] [A1 A2]

TA = [0, 0, 1, 0, 1, 0, 0]

TB = [0, 0, 0, 0, 1, 0, 0]

TQ = [A3]

BQ = []

OQ = [A1, A2, A4, A5, A6, A7]

z = -70225.0

Fragments = [A7 A4 A3 A6 A5 A1] [A2]

TA = [0, 0, 1, 0, 1, 0, 0]

TB = [0, 0, 0, 0, 0, 0, 0]

TQ = [A3, A5]

BQ = []

OQ = [A1, A2, A4, A6, A7]

z = -50625.0

***so here we see in every case z is negative so Vertical Fragmentation not possible.***

**Attribute usage matrix for Cab:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **A1** | **A2** | **A3** | **A4** | **A5** | **A6** |
| **Q1** | **0** | **0** | **0** | **0** | **0** | **0** |
| **Q2** | **1** | **0** | **1** | **1** | **0** | **0** |
| **Q3** | **0** | **0** | **0** | **0** | **0** | **0** |
| **Q4** | **0** | **0** | **0** | **0** | **0** | **0** |
| **Q5** | **0** | **0** | **0** | **0** | **0** | **0** |
| **Q6** | **0** | **0** | **0** | **0** | **0** | **0** |
| **Q7** | **1** | **0** | **0** | **1** | **0** | **0** |

**Attribute affinity matrix for cab:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **A1** | **A2** | **A3** | **A4** | **A5** | **A6** |
| **A1** | **115** | **0** | **40** | **115** | **0** | **0** |
| **A2** | **0** | **0** | **0** | **0** | **0** | **0** |
| **A3** | **40** | **0** | **40** | **40** | **0** | **0** |
| **A4** | **115** | **0** | **40** | **115** | **0** | **0** |
| **A5** | **0** | **0** | **0** | **0** | **0** | **0** |
| **A6** | **0** | **0** | **0** | **0** | **0** | **0** |
| **A7** | **0** | **0** | **0** | **0** | **0** | **0** |

cont( A 0 , A 3 , A 1 ) = 2 \* ( 0 + 10800.0 - 0 ) = 21600.0

cont( A 1 , A 3 , A 2 ) = 2 \* ( 10800.0 + 0.0 - 0.0 ) = 21600.0

cont( A 2 , A 3 , A 4 ) = 2 \* ( 0.0 + 10800.0 - 0.0 ) = 0.0

cont( A 0 , A 4 , A 1 ) = 2 \* ( 0 + 10800.0 - 0 ) = 21600.0

cont( A 1 , A 4 , A 2 ) = 2 \* ( 10800.0 + 28050.0 - 10800.0 ) = 56100.0

cont( A 2 , A 4 , A 3 ) = 2 \* ( 28050.0 + 0.0 - 0.0 ) = 56100.0

cont( A 3 , A 4 , A 5 ) = 2 \* ( 0.0 + 0.0 - 0.0 ) = 0.0

cont( A 0 , A 5 , A 1 ) = 2 \* ( 0 + 0.0 - 0 ) = 0.0

cont( A 1 , A 5 , A 2 ) = 2 \* ( 0.0 + 0.0 - 10800.0 ) = -21600.0

cont( A 2 , A 5 , A 3 ) = 2 \* ( 0.0 + 0.0 - 28050.0 ) = -56100.0

cont( A 3 , A 5 , A 4 ) = 2 \* ( 0.0 + 0.0 - 0.0 ) = 0.0

cont( A 4 , A 5 , A 6 ) = 2 \* ( 0.0 + 0.0 - 0.0 ) = 0.0

cont( A 0 , A 6 , A 1 ) = 2 \* ( 0 + 0.0 - 0 ) = 0.0

cont( A 1 , A 6 , A 2 ) = 2 \* ( 0.0 + 0.0 - 0.0 ) = 0.0

cont( A 2 , A 6 , A 3 ) = 2 \* ( 0.0 + 0.0 - 10800.0 ) = -21600.0

cont( A 3 , A 6 , A 4 ) = 2 \* ( 0.0 + 0.0 - 28050.0 ) = -56100.0

cont( A 4 , A 6 , A 5 ) = 2 \* ( 0.0 + 0.0 - 0.0 ) = 0.0

cont( A 5 , A 6 , A 7 ) = 2 \* ( 0.0 + 0 - 0 ) = 0.0

**Cluster affinity matrix for cab:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **A6** | **A5** | **A3** | **A4** | **A1** | **A2** |
| **A6** | **0** | **0** | **0** | **0** | **0** | **0** |
| **A5** | **0** | **0** | **0** | **0** | **0** | **0** |
| **A3** | **0** | **0** | **40** | **40** | **40** | **0** |
| **A4** | **0** | **0** | **40** | **115** | **115** | **0** |
| **A1** | **0** | **0** | **40** | **115** | **115** | **0** |
| **A2** | **0** | **0** | **0** | **0** | **0** | **0** |

**Partition Algorithm:**

Fragments = [A6] [A5 A3 A4 A1 A2]

TA = [0, 0, 0, 0, 0, 0, 0]

TB = [0, 1, 0, 0, 0, 0, 1]

TQ = []

BQ = [A2, A7]

OQ = [A1, A3, A4, A5, A6]

z = -46225.0

Fragments = [A6 A5] [A3 A4 A1 A2]

TA = [0, 0, 0, 0, 0, 0, 0]

TB = [0, 1, 0, 0, 0, 0, 1]

TQ = []

BQ = [A2, A7]

OQ = [A1, A3, A4, A5, A6]

z = -46225.0

Fragments = [A6 A5 A3] [A4 A1 A2]

TA = [0, 1, 0, 0, 0, 0, 0]

TB = [0, 1, 0, 0, 0, 0, 1]

TQ = []

BQ = [A7]

OQ = [A1, A2, A3, A4, A5, A6]

z = -65025.0

Fragments = [A6 A5 A3 A4] [A1 A2]

TA = [0, 1, 0, 0, 0, 0, 1]

TB = [0, 1, 0, 0, 0, 0, 1]

TQ = []

BQ = []

OQ = [A1, A2, A3, A4, A5, A6, A7]

z = -108900.0

Fragments = [A6 A5 A3 A4 A1] [A2]

TA = [0, 1, 0, 0, 0, 0, 1]

TB = [0, 0, 0, 0, 0, 0, 0]

TQ = [A2, A7]

BQ = []

OQ = [A1, A3, A4, A5, A6]

z = -46225.0

***so here we see in every case z is negative so Vertical Fragmentation not possible.***

**Attribute usage matrix for Owner table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **A1** | **A2** | **A3** | **A4** |
| **Q1** | **1** | **1** | **0** | **1** |
| **Q2** | **0** | **0** | **0** | **0** |
| **Q3** | **0** | **0** | **0** | **0** |
| **Q4** | **0** | **0** | **0** | **0** |
| **Q5** | **0** | **0** | **0** | **0** |
| **Q6** | **0** | **0** | **0** | **0** |
| **Q7** | **0** | **0** | **0** | **0** |

**Attribute affinity matrix for owner:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **A1** | **A2** | **A3** | **A4** |
| **A1** | **30** | **30** | **0** | **30** |
| **A2** | **30** | **30** | **0** | **30** |
| **A3** | **0** | **0** | **0** | **0** |
| **A4** | **30** | **30** | **0** | **30** |

cont( A 0 , A 3 , A 1 ) = 2 \* ( 0 + 0.0 - 0 ) = 0.0

cont( A 1 , A 3 , A 2 ) = 2 \* ( 0.0 + 0.0 - 2700.0 ) = -5400.0

cont( A 2 , A 3 , A 4 ) = 2 \* ( 0.0 + 0.0 - 2700.0 ) = 0.0

cont( A 0 , A 4 , A 1 ) = 2 \* ( 0 + 0.0 - 0 ) = 0.0

cont( A 1 , A 4 , A 2 ) = 2 \* ( 0.0 + 2700.0 - 0.0 ) = 5400.0

cont( A 2 , A 4 , A 3 ) = 2 \* ( 2700.0 + 2700.0 - 2700.0 ) = 5400.0

cont( A 3 , A 4 , A 5 ) = 2 \* ( 2700.0 + 0 - 0 ) = 5400.0

**Cluster Affinity matrix for Owner relation table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **A3** | **A4** | **A1** | **A2** |
| **A3** | **0** | **0** | **0** | **0** |
| **A4** | **0** | **30** | **30** | **30** |
| **A1** | **0** | **30** | **30** | **30** |
| **A2** | **0** | **30** | **30** | **30** |

**Partition algorithm:**

Fragments = [A3] [A4 A1 A2]

TA = [0, 0, 0, 0, 0, 0, 0]

TB = [1, 0, 0, 0, 0, 0, 0]

TQ = []

BQ = [A1]

OQ = [A2, A3, A4, A5, A6, A7]

z = -90000.0

Fragments = [A3 A4] [A1 A2]

TA = [1, 0, 0, 0, 0, 0, 0]

TB = [1, 0, 0, 0, 0, 0, 0]

TQ = []

BQ = []

OQ = [A1, A2, A3, A4, A5, A6, A7]

z = -108900.0

Fragments = [A3 A4 A1] [A2]

TA = [1, 0, 0, 0, 0, 0, 0]

TB = [1, 0, 0, 0, 0, 0, 0]

TQ = []

BQ = []

OQ = [A1, A2, A3, A4, A5, A6, A7]

z = -108900.0

***so here we see in every case z is negative so Vertical Fragmentation not possible.***

There is no any query that uses the attribute related to **cab-driver** table so there is no any need to fragment this table

As we have seen above for the remaining relations Cabs, Driver, User etc where vertical fragmentation has been tried based on the inputs and usage of our queries, I am getting only negative values. So, based on the queries used in this design approach, vertical fragmentation will be done only on the Trip relation. It is fragmented into 2 parts: Trip1 and Trip2.

# Data Allocation & Replication:

As the described applications are all read queries, we will be using local read and remote read time only.

**Time Matrix for Fragments**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Fragment** | **Local Update** | **Remote Update** | **Local Read** | **Remote Read** | **(Remote -**  **Local)** |
| Fragment 1 | 250 | 400 | 200 | 350 | 150 |
| Fragment 2 | 250 | 400 | 200 | 350 | 150 |
| Fragment 3 | 250 | 350 | 200 | 300 | 100 |
| Fragment 4 | 200 | 300 | 150 | 250 | 100 |
| Fragment 5 | 600 | 800 | 575 | 750 | 175 |
| Fragment 6 | 350 | 450 | 300 | 425 | 125 |
| Fragment 7 | 600 | 800 | 575 | 750 | 175 |
| Fragment 8 | 600 | 800 | 575 | 750 | 175 |
| Fragment 9 | 250 | 400 | 200 | 350 | 150 |
| Fragment 10 | 250 | 300 | 150 | 270 | 120 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Transaction** | **Originating Sites** |  | **Frequency** | **Fragment access** |
| Q1 | S2, S3 | 30 |  | F10-1 Read, 2 Write |
| Q2 | S1,S3 | 40 |  | F1-1 Read  F2-1 Read, 2 Write |
| Q3 | S1,S2,S3 | 65 |  | F3-1Read  F4-1 Read, 1 Write |
| Q4 | S1,S2,S4 | 30 |  | F7-1 Read  F8-1 Read, 2 Write |
| Q5 | S2,S3,S4 | 40 |  | F3-1Read  F4-1 Read  F6-1 Read  F7-1 Read  F8-1 Read |
| Q6 | S1,S4 | 50 |  | F5-1 Read, 1 Write  F7-1 Read, 2 Write  F8-1 Read |
| Q7 | S1,S2,S3 | 75 |  | F1-1 Read, 1 Write  F2-1 Read  F7-1 Read  F8-1 Read |

# Redundant All Beneficial Site method:

Redundant all beneficial site method is used for allocating fragments to a particular site. This method operates by calculating both benefit and cost associated with allocating a fragment to a particular site. Based on the values obtained, fragments are allocated.

In our query there is no any Write or update quey so **cost = 0 So, Benefit-cost = Benefit-0**

**= Benefit**

For this method we need,

Remote Read Time - Local Read Time which is (2 \* Propagation Delay ) + Packet -Transmission Time = (2\*60)+2

**= 122**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Fragment | Site | Update Transaction | | # Write \* Freq \* time | Cost (ms) |
| Remote | Local |
| F1 | S1 | T7 from S3, S2 | T7 | (2\*75\*1\*400)+(1\*75\*250) | 78750 |
| S2 | T7 from S1, S3 | T7 | (2\*75\*1\*400)+(1\*75\*250) | 78750 |
| S3 | T7 from S1, S2 | T7 | (2\*75\*1\*400)+(1\*75\*250) | 78750 |
| S4 | T7 from S1,S2,S3 | - | (1\*75\*3)\*400 | 90000 |
| F2 | S1 | T2 from S3 | T2 | (40\*2\*400)+(2\*40\*250) | 52000 |
| S2 | T2 from S1,S3 | - | (2\*40\*2\*400) | 64000 |
| S3 | T3 from S1 | T2 | (40\*2\*400)+(2\*40\*250) | 52000 |
| S4 | T2 from S1,S3 | - | (2\*40\*2\*400) | 64000 |
| F3 | S1 | - | - | - | - |
| S2 | - | - | - | - |
| S3 | - | - | - | - |
| S4 | - | - | - | - |
| F4 | S1 | T3 from S2,S3 | T3 | (65\*1\*2\*300)+(65\*1\*200) | 52000 |
| S2 | T3 from S1,S3 | T3 | (65\*1\*2\*300)+(65\*1\*200) | 52000 |
| S3 | T3 from S2,S1 | T3 | (65\*1\*2\*300)+(65\*1\*200) | 52000 |
| S4 | T3 from S1,S2,S3 | - | (1\*65\*3\*300) | 58500 |
| F5 | S1 | T6 from S4 | T6 | (1\*50\*800)+(1\*50\*600) | 70000 |
| S2 | T6 from S1,S4 | - | (2\*50\*800) | 80000 |
| S3 | T6 from S1,S4 | - | (2\*50\*800) | 80000 |
| S4 | T6 from S1 | T6 | (1\*50\*800)+(1\*50\*600) | 70000 |
| F6 | S1 | - | - | - | - |
| S2 | - | - | - | - |
| S3 | - | - | - | - |
| S4 | - | - | - | - |
| F7 | S1 | T6 from S4 | T6 | (1\*50\*2\*800)+(1\*2\*50\*600) | 140000 |
| S2 | T6 from S1,S4 | - | (2\*1\*50\*2\*800) | 160000 |
| S3 | T6 from S1,S4 | - | (2\*1\*50\*2\*800) | 160000 |
| S4 | T6 from S1 | T6 | (1\*50\*2\*800)+(1\*2\*50\*600) | 140000 |
| F8 | S1 | T4 from S2,S4 | T4 | (2\*30\*2\*800)+(30\*2\*600) | 96000 |
| S2 | T4 from S1,S4 | T4 | (2\*30\*2\*800)+(30\*2\*600) | 96000 |
| S3 | T4 from S1,S2,S3 | - | (3\*30\*2\*800) | 144000 |
| S4 | T4 from S2,S1 | T4 | (2\*30\*2\*800)+(30\*2\*600) | 96000 |
| F9 | S1 | - | - | - | - |
| S2 | - | - | - | - |
| S3 | - | - | - | - |
| S4 | - | - | - | - |
| F10 | S1 | T1 from S2,S3 | - | 2\*2\*30\*300 | 36000 |
| S2 | T1 from S3 | T1 | 1\*2\*30\*200 | 12000 |
| S3 | T1 from S2 | T1 | 1\*2\*30\*200 | 12000 |
| S4 | T1 from S2,S3 | - | 2\*2\*30\*300 | 36000 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Fragment** | **Site** | **Query** | **Benefit** | **Benefit -**  **Cost** |
| F1 | S1 | Q2, Q7 | (1\*40+1\*75)\*122 | -64720 |
| S2 | Q7 | (1\*75)\*122 | -69600 |
| S3 | Q2, Q7 | (1\*40+1\*75)\*122 | -64120 |
| S4 | - | - | -90000 |
| F2 | S1 | Q2, Q7 | (1\*40+1\*75)\*122 | -37970 |
| S2 | Q7 | (1\*75)\*122 | -54850 |
| S3 | Q2, Q7 | (1\*40+1\*75)\*122 | -37970 |
| S4 | - | - | -64000 |
| F3 | S1 | Q3 | (1\*65)\*122 | 7930 |
| S2 | Q3, Q5 | (1\*65+1\*40)\*122 | 12810 |
| S3 | Q3, Q5 | (1\*65+1\*40)\*122 | 12810 |
| S4 | Q5 | (1\*40)\*122 | 4880 |
| F4 | S1 | Q3 | (1\*65)\*122 | -44070 |
| S2 | Q3, Q5 | (1\*65+1\*40)\*122 | -39190 |
| S3 | Q3, Q5 | (1\*65+1\*40)\*122 | -39190 |
| S4 | Q5 | (1\*40)\*122 | -50570 |
| F5 | S1 | Q6 | (1\*50)\*122 | -63900 |
| S2 | - | - | -80000 |
| S3 | - | - | -80000 |
| S4 | Q6 | (1\*50)\*122 | -63900 |
| F6 | S1 | - | - | 0 |
| S2 | Q5 | (1\*40)\*122 | 7930 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| F6 | S3 | Q5 | (1\*40)\*122 | 7930 |
| S4 | Q5 | (1\*40)\*122 | 7930 |
| F7 | S1 | Q4, Q6, Q7 | (1\*30+1\*50+1\*75)\*122 | -121090 |
| S2 | Q4, Q5, Q7 | (1\*30+1\*40+1\*75)\*122 | -142310 |
| S3 | Q5, Q7 | (1\*40+1\*75)\*122 | -145970 |
| S4 | Q4, Q5, Q6 | (1\*30+1\*40+1\*50)\*122 | -125360 |
| F8 | S1 | Q4, Q6, Q7 | (1\*30+1\*50+1\*75)\*122 | -77090 |
| S2 | Q4, Q5, Q7 | (1\*30+1\*40+1\*75)\*122 | -77090 |
| S3 | Q5, Q7 | (1\*40+1\*75)\*122 | -129970 |
| S4 | Q4, Q5, Q6 | (1\*30+1\*40+1\*50)\*122 | -81360 |
| F9 | S1 | - | - | 0 |
| S2 | - | - | 0 |
| S3 | - | - | 0 |
| S4 | - | - | 0 |
| F10 | S1 | - | - | -36000 |
| S2 | Q1 | (1\*30)\*122 | -8340 |
| S3 | Q1 | (1\*30)\*122 | -8340 |
| S4 | - | - | -36000 |

**Fragment Allocation:**

Based on the above values obtained from doing redundant beneficial site method, the fragments are allocated as follows:

|  |  |  |
| --- | --- | --- |
| **Site** |  | **Fragment** |
| **S1** | F2, F5,F7,F8,F9 |  |
| **S2** | F3,F4,F6,F8,F9,F10 |  |
| **S3** | F1,F2,F3,F4,F6,F9,F10 |  |
| **S4** | F5,F6,F9 |  |

● For Fragment 9 there is no query in our assumption so for all sites it’s benefit-cost

value is zero that’s why we will allocate Fragment 9 at any site randomly.

# Physical Design:

Until now we have discussed and designed the database from a logical point of view. Now, with some fixed assumptions we will discuss the physical design of the distributed database. Physical memory primarily constitutes the secondary memory. So, after all our fragmentation and its after processing is done, where do the fragments get stored. Based on their storage, what will be their seek times, rotational latency and the data transfer rate. As said before, following are the assumptions based on which we will be doing the physical design.

❖ Assumptions:

* Fixed length records are considered for all relations.
* The delimiter for each field is the length of the field.
* The total number of records in each fragment will be given below.
* Block size is 1024 bytes.
* A single record does not span over multiple blocks.
* Block pointer is 4 bytes.
* Average seek time is 40 milliseconds irrespective of the sites.
* Average disk rotation time is 20 milliseconds irrespective of the site.
* Block transfer rate is 1 milliseconds irrespective of the site.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Fragment** | **Relations** | **No of Records** | **Single**  **Record**  **Size** | **Total Size** | **Blocking**  **Factor** | **No of Blocks** |
| Fragment 1 | Cab 1 | 200 | 30 | 6000 | 27 | 8 |
| Fragment 2 | Cab 2 | 200 | 30 | 7500 | 27 | 10 |
| Fragment 3 | Driver 1 | 150 | 30 | 4500 | 31 | 5 |
| Fragment 4 | Driver 2 | 100 | 30 | 3000 | 31 | 4 |
| Fragment 5 | Payment | 5000 | 50 | 250000 | 23 | 218 |
| Fragment 6 | User | 2000 | 40 | 80000 | 25 | 80 |
| Fragment 7 | Trip 1 | 5000 | 30 | 150000 | 37 | 136 |
| Fragment 8 | Trip 2 | 5000 | 50 | 250000 | 37 | 136 |
| Fragment 9 | Cab-Driver | 300 | 20 | 6000 | 31 | 10 |
| Fragment 10 | Owner | 150 | 20 | 3000 | 27 | 6 |

Considering the assumption, we can easily calculate the size of a single record (tuple) of every relation with the help of Global Schema. The above table gives the number of records in each relation, size of each record, blocking factor for a particular block of that relation and number of blocks required to store the entire relation.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Fragment** | **Relations** | **Indexing Type** | **Indexing**  **Attribute** | **Is it a Key?** |
| Fragment 1 | Cab 1 | Clustered | Type | No |
| Fragment 2 | Cab 2 | Clustered | Type | No |
| Fragment 3 | Driver 1 | Clustered | Rating | No |
| Fragment 4 | Driver 2 | Clustered | Rating | No |
| Fragment 5 | Payment | Primary | Ref\_id | Yes |
| Fragment 6 | User | Primary | User\_id | Yes |
| Fragment 7 | Trip 1 | Primary | Trip\_id | yes |
| Fragment 8 | Trip 2 | Primary | Trip\_id | Yes |
| Fragment 9 | Cab-Driver | Primary | Cab\_id, D\_id | Yes |
| Fragment 10 | Owner | Primary | Owner\_id | Yes |

In some queries, we use range to filter out records. Therefore, using B+ tree would benefit in access time.Some fragments are being accessed by point queries, which can be efficient in physical access using B tree. The following table explains what is the disk block access time to extract a particular record for all the relations.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Fragment** | **Relations** | **No of Records** | **No of Blocks** | **Index size per record** | **Index blocking factor** | **No of index block** | **No of block**  **access with indexing** | **No of block**  **access**  **without indexing** |
| Fragment 1 | Cab 1 | 200 | 8 | 4+10 | 73 | 3 | 3 | 8 |
| Fragment 2 | Cab 2 | 200 | 10 | 4+10 | 73 | 3 | 3 | 10 |
| Fragment 3 | Driver 1 | 150 | 5 | 4+10 | 73 | 3 | 3 | 5 |
| Fragment 4 | Driver 2 | 100 | 4 | 4+10 | 73 | 2 | 2 | 4 |
| Fragment 5 | Payment | 5000 | 218 | 4+4 | 128 | 40 | 7 | 218 |
| Fragment 6 | User | 2000 | 80 | 4+4 | 128 | 16 | 5 | 80 |
| Fragment 7 | Trip 1 | 5000 | 136 | 4+4 | 128 | 40 | 7 | 136 |
| Fragment 8 | Trip 2 | 5000 | 136 | 4+4 | 128 | 40 | 7 | 136 |
| Fragment 9 | Cab-Driver | 300 | 10 | 4+4 | 128 | 3 | 3 | 10 |
| Fragment 10 | Owner | 150 | 6 | 4+4 | 128 | 2 | 2 | 6 |

The above table discusses the number of access each fragment takes in case, if all the fragments are ordered and indexing exists on top of all of them. This shows that indexing drastically reduces the number of block accesses required to find a record present in a table. So, the usage of trees for indexing is justified.

**Access time to Local Query:**

Next, we will calculate the time taken to query each relation considering the relation is locally present. Even though in our sample SQL’s are just read still provided the Local (keyword local because it’s not yet distributed) Query Time and Local Update Time formulae

1. Local Query Time = (Seek Time(40)+ Latency(20) + Block Transfer Time(0.5))

\* N

2. Local Update Time = (Seek Time + Latency + Block Transfer Time) \* N\*2 Where:

* N is number of disk block access, which depends on the relation (we already calculated this above and will consider indexed logic # of block access).
* \*2 is included in the Update time, since the data block has to be fetched into memory from the disk, updated and then written back to the disk.

**Access time to remote query:**

Let us consider the distance between sites. Assume that each site is located at some distance say 1200kms from the other site and the speed of the transmission media connecting the sites is 2\*(10^6) meters/second.

Propagation delay between the sites is computed as below.

* Propagation delay = (Distance between sites)/ (Speed of Transmission media)

= 120 \* 10^3 / 2 \* 10^6

= 0.06s

=60 milliseconds.

Let us assume bandwidth of the network as 1MBps and data is exchanged between sites in the form of packets. Package size is assumed to be 2000 bytes. Transmission Time for a packet is given by,

Packet Transmission Time = (Size of packet) / Bandwidth

= (2000 B) / (10^6 B/s)

= 0.002s

= 2 milliseconds

* **Remote Query Time** = Local Query Time + 2 \* Propagation Delay + Packet

Transmission Time

* **Remote Update Time** = Local Update Time + 2 \* Propagation Delay

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Fragment** | **Relations** | **No of**  **Records** | **No of**  **Blocks** | **No of index block access** | **Local query time(ms)** | **Remote query time(ms)** |
| Fragment 1 | Cab 1 | 200 | 8 | 3 | 183 | 305 |
| Fragment 2 | Cab 2 | 200 | 10 | 3 | 183 | 305 |
| Fragment 3 | Driver 1 | 150 | 5 | 3 | 183 | 305 |
| Fragment 4 | Driver 2 | 100 | 4 | 2 | 122 | 244 |
| Fragment 5 | Payment | 5000 | 218 | 40 | 2440 | 2562 |
| Fragment 6 | User | 2000 | 80 | 16 | 976 | 1098 |
| Fragment 7 | Trip 1 | 5000 | 136 | 40 | 2440 | 2562 |
| Fragment 8 | Trip 2 | 5000 | 136 | 40 | 2440 | 2562 |
| Fragment 9 | Cab-Driver | 300 | 10 | 3 | 183 | 305 |
| Fragment 10 | Owner | 150 | 6 | 2 | 122 | 244 |

# Work Area Space and System specification:

In work area space, we would like to measure the maximum amount of buffer space required for computation.

Let’s assume that the largest 4 fragments in our whole database contained 5000, 2000, 5000, 5000 records each. In a situation where we are required to operate on these four fragments, in the worst case we will be required 17,000 records.

The assumed maximum record size is 50 Bytes.

Total Maximum Buffer Size : 17,000 \* 50 Bytes = **850,000 Bytes.**

For the disk space, as we will be storing all our relations in the disk, we will have to combine the size of all the relations to calculate total required space.

So, the total disk space required would be:

6000 + 7500 + 4500 + 3000 + 250000 + 80000 + 150000 + 250000 + 6000 + 3000 = 760,000 Bytes

**A disk containing roughly 760 KB free space would suffice for our database.**