**NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA**

**Department of Information Technology**

Advanced Database Systems

Assignment 1

**TOURSIM AGENCY DATABASE MANAGEMENT SYSTEM**

Name : Rathinaraja Jeyaraj

Roll No : IT15F01

Class : PhD 1st semester

**Contents**

1. Problem Description 03

2. Actors 05

3. Sample queries 05

4. Entities that are maintained in different locations 05

5. Conceptual Model (EER) 06

6. Global Conceptual Schema 07

7. Normalization 08

8. Global schema 17

9. Fragmentation 19

10. Physical Design 22

11. Allocation and Replication 28

**1. PROBLEM STATEMENT:**

Tourism has been growing nowadays as a trademark for a country to attract foreign visitors and thereby increasing the revenue. As number of countries and their proposed places for visit are gradually growing since it plays a vital role to show the country reputation and increase revenue. On the other hand, every tourists wishes to get better information about the tourist spot, travel and accommodation facility, preferably, in less expense. Therefore, it is necessary to maintain all the information about tourist’s spots that are geographically located and the related travel and accommodation information in the respective locations to help the visitors to make a better trip. There are many third party companies who offer this as a service based on this idea to provide better service on the palm. This project aims designing an information system including airtravel information’s and hotel availability in the appropriate places to pin point to avoid receding to multiple sites. A third party company named “international tourism management” comes forward to make a better tie-up with several travel agencies, hotels and tourism information for offering a better choice to the visitors. Therefore, it is required to make a good database design to the company. Their requirement is as follows.

Tourism agency provides portal for passengers to explore their trip. Agency provides its details such as government authenticated id for identification, name, address, contact number and customer support for passenger. In order to use this service a passenger is ought to register with the tourism agent by providing following details: passport number for verification, name has first name and last name, sex, date of birth, nationalities (maybe more than one), address (comprises present and permanent address) and their present contact number. Age can be obtained from the date of birth. This tourism agency may have one or more passengers registered to book. Once they registered, they are able to get information about the tourist sites which are in multiple locations in a country along with the hotel availability nearby and airline travel facility.

This tourist agent must have at least one or more tourist’s spot information and hotels information.Information’s about tourist spot are location number,name, location, country, nearby places and hotels to help passenger to have more sites to visit, visiting time for making passenger’s to be prompt to have their visit before closing, current weather that helps passengers to be pre-cautious from extreme cold or hot, risk factor that specifies the safety of the place. Tourism agency must have at least one site's information and every sitesshould have to be registered to the agent. Hotels offer various type of stay such as A/C, non A/C, and dormitory etc. for various rents. further information about the hotels are obtained if only if they have properly registered in the agency with their license, location, address and contact number. Agent has more than one hotels registered. Every hotel may or may not have a tourist spot nearby and every site may optionally have number of hotels nearby.

The tourism agent provides only air travel facility with more than one airline. It provides the following information about every airlines: flight number, name, date on which to make trip, departure, arrival, travel type that includes business and economy class, and its fare. On surfing about airlines for trip, passengermay make more than one schedule for booking according to their wish. Every schedule must belong toany one of the airlines. Schedule includes information such as schedule id, departure time, arrival time, total travel time and type of catering depending on the travel type including the flight details.Once a passenger confirmed their visiting spot, airlines, hotel they may proceed for booking. Booking details are booking id, status whether confirmed with the entire schedule, cancel that allows cancelling the schedule and reference to the travel, hotel, site details. Booking may be done with some discount to attract customers. To confirm booking, transactionsmay be done throughany bank throughdifferent modes of payment like DD, online, etc,. total amount to be paid is calculated considering the discount and payment. Acknowledgement is generated and updated in the user account and send to the passenger's mobile and email contact. These are the minimum requirements for an agency to maintain information system.

However, all these information’s are not maintained in a single location because different sector maintains their own information system. Therefore, distributed scenario comes into picture that allows control and dataflow between multiple sites to share information. Based on the requirement, four sites are identified. The sites are (i) travel agencyorganizes airline information management system, (ii) tourism agentmanages passenger, booking and payment transaction details, (iii) hotel management keeps track of all licensed hotels, and (iv) government tourism management has details about various tourist spots.

**2. ACTORS:** people who interact with the database

* passengers
* travel agency

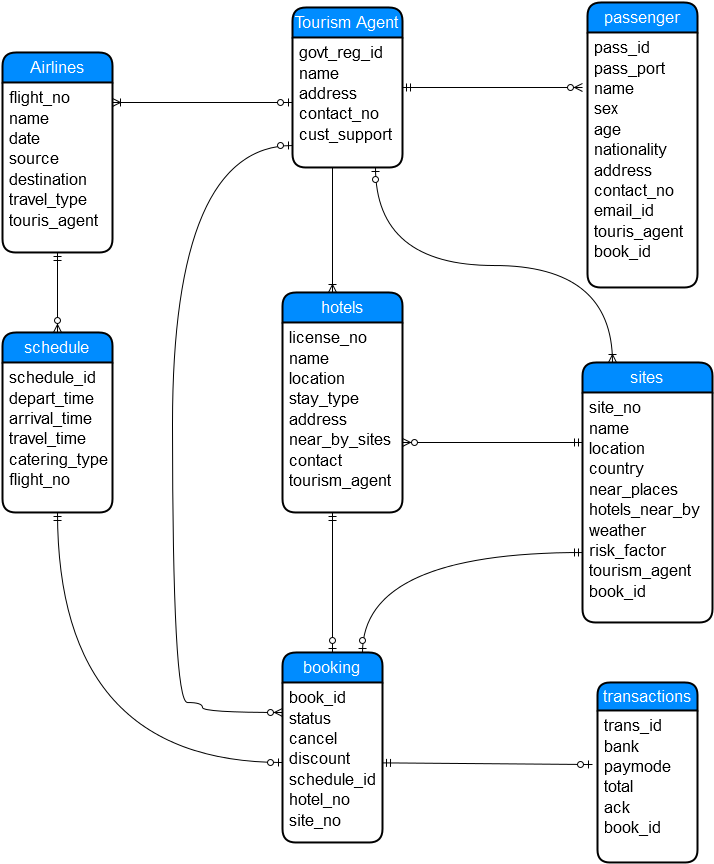
**3. SOME SAMPLE QUERIES:**

* A passenger can
  + check list of tourist places in a country
  + obtain list of flights that fly from one place to another place.
  + get list of flights that reach destination in less than 10 hours.
  + find list of hotels that are near by a tourist place.
  + get the information about booking status
* Tourism agency can
  + get list of passengers who are from same country
  + track a booking details of a specified passenger.
  + verify the transactions of a particular visitor
  + get email-id of all passengers to send discount and offer detail for a particular site
  + obtain list of airlines that have been registered with them.

**4. ENTITIES THAT ARE MAINTAINED IN DIFFERENT LOCATIONS:**

|  |  |  |
| --- | --- | --- |
| **ENTITIES** | **LOCATION** | **INFORMATION** |
| Tourism agent, passenger, booking, transaction | Site A | Provides information about the agency, passenger details, and their booking and transfer information. |
| Tourist site information | Site B | Mostly government manages this part as it has control to declare a place as tourist spot considering such as risk factor etc. |
| Airline agency | Site C | airline agency provides information about Flights, running between various tourist spots. |
| Hotel management | Site D | Provides various resident facility at the tourist spots with various type of accommodation |

**5. EER MODEL (CONCEPTUAL MODEL)**

****

**6. GLOBAL CONCEPTUAL SCHEMA:**

The transformation from the entity-relationship model to the relational model is very straightforward. A feasible set of relational schemas is as follows.

**Tourism\_Agent:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Govt\_reg Id | name | address | Contact\_no | Cust\_support |

**Passanger:**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pass\_id | Pass\_no | name | sex | dob | age | nationality | address | contact | mail | Tour\_agent |

**Booking:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Book\_id | status | cancel | discount | Hotel\_no | Site\_no | schd\_id | Tour\_agent |

**Transactions:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Trans\_id | bank | Paymode | total | ack | Book\_id |

**Airlines:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Flight no | Name | date | source | destination | Travel\_type | Tour\_agent |

**Schedule:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Schedule\_id | Dep\_time | Arrival\_time | Trave\_time | Cater\_type | Flight\_no |

**Hotel:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| license | name | Location | Type\_of\_stay | Address | Near\_sites | contact | Tour\_agent |

**Sites:**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Site\_no | Name | Location | Country | Near\_ places | Hotels\_near | Weather | Risk\_factor | Book\_id | Tour\_agent |

**7. NORMALIZATION:**

Some queries might have to travel with more than one table based on foreign key by some kind of joining. If we have 100’s of table then joining kind of operation will take a lot of time. So, it is undesirable. Alternative solution is keeping a universal or central table having all attributes together. It eases our information retrieval but there can be lots duplication or redundant values. **Redundancy** is repeated values in a same table, which leads to wastage of space, example, for 100 employees working for a same department, we have to repeat same department information for all 100 employees in a table. We have huge storage capacity at lower cost nowadays. So, what is the trouble here? anomalies.

**Anomalies** is a kind of inconsistent information and overhead. Database won’t show any error and will simply accept the values. It results to imprecise output, example, from 100 employees in “sales” department, I changed department name to “production” by mistake for 10 employees. Now if I query for list of employees in sales department, it will show only 90 employees details, 10 are left out. Database won’t show any error for changing 10 employees detail but it simply accepts. Such kind of inconsistency is called anomaly. It can happen while insert, delete and updatinginformation.

1. **Insert anomaly:** while inserting employee information we might enter wrong department details, which will lead to anomaly.
2. **Delete anomaly:** there are 100 employees in a department “sales”. If we delete all 100 employees in that department then we would lose department details too. There is no other way to extract department details of “sales”.
3. **Update anomaly:** while updating a table we might enter some other details by mistake, which will lead to anomaly.

These are the overheads, when you combine all tables into a centralized one. So, how can we eliminate them? Solution is dividing tables as smallest as possible. But, how small a table should be? To the level of anomalies wont occurs, ideally table having 2 attributes. But, achieving this will increase querying time. Therefore, how to determine the number of attributes to keep in table which won’t lead to large querying time and anomalies?

Idea is dividing large table into small tables having less number of attributes in such a way that your design reduces anomalies and subsequently degree of redundancy that are present in the table. This systematic procedure is called **normalization**.

Normalization is carried out having functional dependency and candidate keys in mind. It is a step-wise process that divides relations into several pieces until we eliminate redundancy and anomalies.

There are many steps to achieve this normalization.

1. 1st normal form
2. 2nd normal form
3. 3rd normal form
4. Boyce-codd normal form

Each follows their own set of rules. Let’s discuss with the help of tourism agency databse management system’s global conceptual schema.

**What is key in general?** Given a value of an attribute, we must be able to uniquely identify all other attributes given in a table. That is, a complete row also called tuple or record.

|  |  |  |
| --- | --- | --- |
| A | B | C |
| 1 | c | d |
| 2 | a | b |
| 3 | e | f |

A 🡪 BC

🡪This symbols denotes “determines” or “returns”

(group of attributes) 🡪 (group of attributes)

LHS is called key and RHS is called values.

Example, if I say 3 in the attribute A, then it must return BC values.🡪is called functional dependency. The term “functional” implies group of attributes that determine another group of attributes. We can also say attribute level dependency instead of functional dependency. But, mathematically speaking this kind of representation (LHS 🡪 RHS) is called **“functionally dependent”.**Therefore, applying mathematics (set theory) on table design helps us dealing attributes. There are 3 different kinds of FD’s.

1. **Trivial Functional Dependency (FD):**  What is got on RHS is already LHS

Example: A determines itself.

A🡪A

A🡪 AB

AB🡪A

In general form to represent this . however, there are no useful operation using this kind of dependency.

1. **Non-trivial FD:** given a value of an attribute get a unique value.

Example:

A 🡪 B

A 🡪 BC

AB 🡪 CD

In general, the rule is

1. **Semi non-trivial FD:** it is not deriving completely new information. It partially brings a record, that is, not the value of entire set of attributes.

Example:

AB 🡪 BC B is common on both sides

ACD 🡪 EFCD CD is common both sides

It is generally represented as

FD’s are quite useful in identifying keys, identifying equivalences of FS’s and finding minimal FD set. There are rules applied on FD such as inference rules (reflexive, transitivity, augmentation, union, etc), closure.

**Closure** – how many attributes you are able to determine by one attribute. It is the base of entire normalization process. Using closure property we can determine candidate keys.

**Candidate keys** are key or set of keys that identify all other attributes in a table. A table can have many candidate keys, but at any moment, only one candidate key can be as a primary key of a table. If there are n attributes then candidate keys are possible except null.

Sometimes, candidate key with non-key attribute uniquely determine a table. Such key is called **superkey.**

Minimal super key or candidate key which has less no of attributes is called **primary key,** which uniquely identifies a tuple.

In FD’s, LHS must be a key for every table. In BCNF, we have 0% redundancy in table. To achieve this, we go through series of normalization from 1st NF to BCNF. It is not mandatory to go from 1st NF to BCNF. But, it is a convention to follow this sequence.

Every normal form should be lossless, and FD preserved

**1st Normal Form:** A relation is said to be in first normal form then it should satisfy the following

* No multi-valued attribute
* No composite attribute
* identify primary key

Here, the relationship is converted to either relation or foreign key or merging relations.

**Foreign key:** giving primary key of one table as a reference to another table.

**Tourism\_Agent:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Govt\_reg Id | name | address | Contact\_no | Cust\_support |

**Passanger:**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pass\_id | Pass\_no | name | sex | dob | age | address | contact | mail | Tour\_agent |

**nationality:**

|  |  |
| --- | --- |
| Pass\_id | country |
| 1 | india |
| 1 | USA |
| 2 | germany |

**Booking:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Book\_id | status | cancel | discount | Hotel\_no | Site\_no | schd\_id | Tour\_agent |

**Transactions:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Trans\_id | bank | Paymode | total | ack | Book\_id |

**Airlines:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Flight no | Name | date | source | destination | Travel\_type | Tour\_agent |

**Schedule:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Schedule\_id | Dep\_time | Arrival\_time | Trave\_time | Cater\_type | Flight\_no |

**Hotel:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| license | name | Location | Address | Near\_sites | contact | Tour\_agent |

**Near\_by\_sites:**

|  |  |  |
| --- | --- | --- |
| Hotel id | place | Site\_no |
| 1 | xxx | 2 |
| 2 | yyy | 1 |
| 1 | zzzz | 1 |

**Sites:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Site\_no | Name | Location | Country | Weather | Risk\_factor | Book\_id | Tour\_agent |

**Outcome of 1st normalization:**

* Primary key has been identified in each table using closure property (minimal super key)
* Composite attributes has been resolved
* Multi-valued attributes has been resolved

**2nd Normal Form:**

1. Repeating column values are taken out and maintained in a separate table. So that change can be done only once in the new table rather than all entries in the first table. Rule is foreign key must be on the N side else again multi-value in a column will occur.
2. Identify **prime attribute** (part of candidate key that determines anything else), it is also called **partial dependency**, and eliminate it. Because, 2nd NF is based on **Full Functional dependency** (key should determine all other attributes in a table)
3. Use foreign key on many side

**Tourism\_Agent:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Govt\_reg Id | name | address | Contact\_no | Cust\_support |

**Passanger:**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pass\_id | Pass\_no | name | sex | dob | age | address | contact | mail | Tour\_agent |

**nationality:**

|  |  |
| --- | --- |
| Pass\_id | country |
| 1 | india |
| 1 | USA |
| 2 | germany |

**Booking:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Book\_id | status | cancel | Hotel\_no | Site\_no | schd\_id | Tour\_agent |

**percentage:**

|  |  |
| --- | --- |
| Book id | Discount (%) |
| 1 | 10 |
| 2 | 8 |
| 3 | 12 |

**Transactions:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Trans\_id | bank | Paymode | total | ack | Book\_id |

**Airlines:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Flight no | Name | date | source | destination | Travel\_type | Tour\_agent |

**Schedule:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Schedule\_id | Dep\_time | Arrival\_time | Trave\_time | Flight\_no |

**menu:**

|  |  |
| --- | --- |
| Schedule\_id | Cater\_item |
| 1 | rice |
| 2 | pizza |
| 3 | dhaal |

**Hotel:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| license | name | Location | Address | Near\_sites | contact | Tour\_agent |

**Type\_of\_stay:**

|  |  |
| --- | --- |
| Hotel id | type |
| 1 | AC |
| 1 | Non-AC |
| 1 | dormatory |

**Near\_by\_sites:**

|  |  |  |
| --- | --- | --- |
| Hotel id | place | Site\_no |
| 1 | xxx | 2 |
| 2 | yyy | 1 |
| 1 | zzzz | 1 |

**Sites:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Site\_no | Name | Location | Country | Weather | Risk\_factor | Book\_id | Tour\_agent |

**Risk\_factor:**

|  |  |
| --- | --- |
| Site\_no | % |
| 1 | 5 |
| 2 | 2 |
| 3 | 10 |

**3rd Normal Form:**

* Only columns with direct dependency of the primary key shall be in the entity.
* No transitive dependencies: non-prime attributes transitively depending on the key. . Example: A🡪B🡪C == A 🡪 C.

A 🡪 BB is non-key attribute here

B🡪C suddenly becomes key attribute here. because of this, we will get repeated values in a column. Therefore, it should be eliminated.

* 3rd NF should hold the condition that: if X 🡪 Y then

Either X is a super key

Or Y is a prime attribute

Following this condition will never allow transitive dependency.

**Tourism\_Agent:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Govt\_reg Id | name | address | Contact\_no | Cust\_support |

**Passanger:**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pass\_id | Pass\_no | name | sex | dob | age | address | contact | mail | Tour\_agent |

**nationality:**

|  |  |
| --- | --- |
| Pass\_id | country |
| 1 | India |
| 1 | USA |
| 2 | germany |

**Booking:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Book\_id | status | cancel | Hotel\_no | Site\_no | schd\_id | Tour\_agent |

**percentage:**

|  |  |
| --- | --- |
| Book id | Discount (%) |
| 1 | 10 |
| 2 | 8 |
| 3 | 12 |

**Transactions:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Trans\_id | bank | Paymode | total | ack | Book\_id |

**Airlines:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Flight no | Name | date | source | destination | Travel\_type | Tour\_agent |

**Schedule:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Schedule\_id | Dep\_time | Arrival\_time | Trave\_time | Flight\_no |

**menu:**

|  |  |
| --- | --- |
| Schedule\_id | Cater\_item |
| 1 | rice |
| 2 | pizza |
| 3 | dhaal |

**Hotel:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| License | name | Location | Address | Near\_sites | contact | Tour\_agent |

**Type\_of\_stay:**

|  |  |
| --- | --- |
| Hotel id | type |
| 1 | AC |
| 1 | Non-AC |
| 1 | dormatory |

**Near\_by\_sites:**

|  |  |  |
| --- | --- | --- |
| Hotel id | place | Site\_no |
| 1 | xxx | 2 |
| 2 | yyy | 1 |
| 1 | zzzz | 1 |

**Sites:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Site\_no | Name | Location | Country | Weather | Risk\_factor | Book\_id | Tour\_agent |

**Risk\_factor:**

|  |  |
| --- | --- |
| Site\_no | % |
| 1 | 5 |
| 2 | 2 |
| 3 | 10 |

Every candidate key in a table determines all other attributes and no non-key attributes determine any attributes in the tables.

4th NF is not required since we eliminate repeated values in the 2NF itself.

**BCNF:**

Every 3NF is not BCNF but if a table is in BCNF then it is already in 3NF.

**BCNF** says every LHS of FD’s must be the key of one of the tables. That is, every prime attribute should determine all other attributes in a table. Therefore, the above tables satisfy BCNF.

**8. GLOBAL SCHEMA:**

|  |  |
| --- | --- |
| **Tourism\_agent** | |
| **Attribute name** | **Attribute size (type in bytes)** |
| Govt\_reg Id | Int(2) |
| Name | char(10) |
| address | Char(20) |
| Contact\_no | Int(2) |
| Cust\_support | Int(2) |

|  |  |
| --- | --- |
| **passenger** | |
| **Attribute name** | **Attribute size (type in bytes)** |
| Pass\_id | Int(2) |
| Pass\_no | Int(2) |
| name | Char(10) |
| sex | Int(2) |
| dob | Date(3) |
| age | Int(2) |
| address | Char(20) |
| contact | Int(2) |
| mail | Char(20) |
| Tour\_agent | Int(2) |

|  |  |
| --- | --- |
| **percentage** | |
| **Attribute name** | **Attribute size (type in bytes)** |
| Book\_id | Int(2) |
| Discount (%) | int(2) |

|  |  |
| --- | --- |
| **nationality** | |
| **Attribute name** | **Attribute size (type in bytes)** |
| Pass\_id | Int(2) |
| country | char(4) |

|  |  |
| --- | --- |
| **transactions** | |
| **Attribute name** | **Attribute size (type in bytes)** |
| Trans\_id | Int(2) |
| bank | char(10) |
| paymode | Char(10) |
| ack | Char(4) |
| Book\_id | Int(2) |

|  |  |
| --- | --- |
| **booking** | |
| **Attribute name** | **Attribute size (type in bytes)** |
| Book\_id | Int(2) |
| status | char(2) |
| cancel | Char(2) |
| hotel\_no | Int(2) |
| Site\_no | Int(2) |
| Schd\_id | Int(2) |
| Tour\_agent | Int(2) |

|  |  |
| --- | --- |
| **flight** | |
| **Attribute name** | **Attribute size (type in bytes)** |
| Flight no | Int(2) |
| Name | char(10) |
| date | Date(5) |
| source | char(10) |
| destination | char(10) |
| Travel\_type | char(10) |
| Tour\_agent | Int(2) |

|  |  |
| --- | --- |
| **schedule** | |
| **Attribute name** | **Attribute size (type in bytes)** |
| Schedule\_id | Int(2) |
| Dep\_time | Int(2) |
| Arrival\_time | Int(2) |
| Trave\_time | Int(2) |
| Flight\_no | Int(2) |

|  |  |
| --- | --- |
| **Type of stay** | |
| **Attribute name** | **Attribute size (type in bytes)** |
| Hotel id | Int(2) |
| type | char(10) |

|  |  |
| --- | --- |
| **Near\_by\_sites** | |
| **Attribute name** | **Attribute size (type in bytes)** |
| Hotel id | Int(2) |
| place | char(10) |
| Site\_no | Int(2) |

|  |  |
| --- | --- |
| **menu** | |
| **Attribute name** | **Attribute size (type in bytes)** |
| Schedule\_id | Int(2) |
| Cater\_item | char(20) |

|  |  |
| --- | --- |
| **hotel** | |
| **Attribute name** | **Attribute size (type in bytes)** |
| License | Int(2) |
| name | char(20) |
| location | char(20) |
| address | char(20) |
| Near\_sites | char(20) |
| contact | Int(2) |
| Tour\_agent | Int(2) |

|  |  |
| --- | --- |
| **site** | |
| **Attribute name** | **Attribute size (type in bytes)** |
| Site\_no | Int(2) |
| name | char(10) |
| location | char(10) |
| country | char(10) |
| weather | char(10) |
| Risk\_factor | Int(2) |
| Book\_id | Int(2) |
| Tour\_agnet | Int(2) |

|  |  |
| --- | --- |
| **Risk\_factor** | |
| **Attribute name** | **Attribute size (type in bytes)** |
| Site\_no | Int(2) |
| % | Int(2) |

**9. FRAGMENTATION:**

Database tables are usually decomposed into smaller fragments for following reasons:

1. when storage exhausted out
2. for parallel processing
3. for load balancing
4. to improve query response time
5. for better local processing
6. availability

decomposed fragments are placed into some other site to facilitate query and optimize other quality of services. These fragments permit number of transactions concurrently. Taking copy of a relation and maintaining 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:

1. **completeness**: all row or column must be present in at least one site.
2. **Reconstruction:** while reconstructing the relation, there should not be any inconsistency or loss of data.
3. **Disjointness:** 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 tourism management system.

1. **check list of tourist places in a country.**

SELECT name

FROM sites

WHERE country=”GERMANY”

1. **obtain list of flights that fly from one place to another place.**

SELECT name, date

FROM Airlines

WHERE source=”newyork” and destination=”melbourne”

1. **get list of flights that reach destination in less than 10 hours.**

SELECT a.name, a.date

FROM Airlines a, Schedule s

WHERE a.source=”newyork” and a.destination=”melbourne” and s.traveltime<=10

1. **find list of hotels that are near by a tourist place.**

SELECT s.hotels\_near\_by, h.name, h.location

FROM sites s, hotels h

WHERE s.name=”GERMANY”

1. **get the information about booking status for a passenger.**

SELECT pass\_id, name, status

FROM booking, passenger

WHEREbook\_id=23 and pass\_id=90

1. **get list of passengers who are from particular country.**

SELECT pass\_id, name

FROM passenger

WHERE country=”india”

1. **track a booking details of a specified passenger.**

SELECTpass\_id, status, discount

FROM passenger, booking

WHEREpass\_id= 012

1. **find the transactions of a particular visitor**

SELECT \*

FROM transaction, passenger

WHEREpass\_id= (select \* from passenger where transaction.bookid = passenger.bookid )

1. **get email-id of all passengers**

SELECTemail\_id

FROM passenger

1. **obtain list of airlines that have been registered with tourist agent**

SELECT name, tourism\_agent.name

FROM airlines, tourism\_agent

WHEREairlines.tourism\_agent=govt\_reg\_id

**9.1 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 doing it. Primary and derived horizontal fragmentation. But, it is usually done using the predicate defined on the queries. In the above listed queries, the most frequently used query are first and 4th query. I have chosen these queries because the 90% of passenger searches for a good tourist spot which maintains pleasant weather to spend the day cool. Hotels are another major concern for the visitors to be sure about safety and less expensive. Therefore, based on the frequency of the queries, the predicates are

(i) country=”value” & weather=”range” from site relation

site1= SELECT \* FROM site WHERE (country=”INDIA” and weather=35)

site2= SELECT \* FROM site WHERE (country =”NEW ZEALAND” and weather=7)

site3= SELECT \* FROM site WHERE (country =”GREENLAND” and weather= -5)

site4= SELECT \* FROM site WHERE (country =”USA” and weather=10)

site5= SELECT \* FROM site WHERE (country =”INDIA” and 36<weather<40)

site6= SELECT \* FROM site WHERE (country =”NEW ZEALAND” and 7<weather<10)

site7= SELECT \* FROM site WHERE (country =”GREENLAND” and -6<weather<-5)

site8= SELECT \* FROM site WHERE (country =”USA” and 10<weather<15)

(ii) Location = “value” from hotel relation

Location 1= SELECT \* FROM hotels WHERE (location= “INDIA”)

Location 2= SELECT \* FROM hotels WHERE (location= “NEW ZEALAND”)

Location 3= SELECT \* FROM hotels WHERE (location= “GREENLAND”)

Location 4= SELECT \* FROM hotels WHERE (location= “USA”)

**9.2 Vertical Fragmentation:**

The vertical fragmentation of a relation R produces subschemas R1, R2, R2,…Rn. Each of which contains subset of attributes, and only one fragment has 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:

1. **grouping (bottom up):** done by combining every two attributes at a time and takes a long time if number of attributes are over 100 to get desired fragments.
2. **splitting (top down) :** given all attributes together is taken as a fragment and split them as many fragments as you want to get. This is much quicker than the first method.

[**note**: all calculations are attached in a A4 sheets].

The list of vertical fragments are: based on the vertical fragmentation calculations, there are no possibilities and benefits in making vertical fragmentation.

**Other fragments are:**tourism\_agent, Airlines, Schedule, booking, passenger, transaction, nationality, percentage, menu, type\_of\_stay, near\_by\_sites, and risk\_factor

**10. PHYSICAL DESIGN:**

This physical design talks about how these fragments are stores in secondary memory. Based on global schema defined in Section 6, the size of all the attributes of all relations remains same. Following are the assumptions which are considered for the physical design:

* Fixed length records are considered for all relations.
* The delimiter for each field is length of the field
* Total number of records in respective relations (provided in below table).
* Block size is 1024 bytes.
* Record doesn’t span over multiple blocks (this can be achieved by taking floor function during calculating number of records per block to restrict single record doesn’t span over blocks).
* Block pointer(Bp) size is 4 bytes
* Average Seek Time(S) is 20 ms irrespective of any site.
* Average Disk rotation time (Latency) Time (L) is 10 ms irrespective of any site.
* Block transfer rate (Tr) is 0.5 ms irrespective of any site.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Fragment** | **Relation** | **# of records** | **Record size in bytes** | **Blocking factor** | **# no of blocks** |
| Fragment 1 | Site 1 | 300 | 48 | 21 | 15 |
| Fragment 2 | Site 2 | 200 | 48 | 21 | 10 |
| Fragment 3 | Site 3 | 400 | 48 | 21 | 19 |
| Fragment 4 | Site 4 | 200 | 48 | 21 | 10 |
| Fragment 5 | Site 5 | 150 | 48 | 21 | 8 |
| Fragment 6 | Site 6 | 200 | 48 | 21 | 10 |
| Fragment 7 | Site 7 | 100 | 48 | 21 | 5 |
| Fragment 8 | Site 8 | 300 | 48 | 21 | 15 |
| Fragment 9 | hotel 1 | 500 | 86 | 11 | 46 |
| Fragment 10 | hotel 2 | 700 | 86 | 11 | 64 |
| Fragment 11 | hotel 3 | 200 | 86 | 11 | 19 |
| Fragment 12 | hotel 4 | 400 | 86 | 11 | 37 |
| Fragment 13 | tourism\_agent | 200 | 36 | 28 | 8 |
| Fragment 14 | Airlines | 1000 | 49 | 20 | 50 |
| Fragment 15 | Schedule | 7000 | 10 | 102 | 69 |
| Fragment 16 | Booking | 1200 | 14 | 73 | 17 |
| Fragment 17 | Passenger | 5000 | 65 | 15 | 334 |
| Fragment 18 | transaction | 8000 | 28 | 36 | 223 |
| Fragment 19 | Nationality | 100 | 6 | 170 | 1 |
| Fragment 20 | percentage | 200 | 4 | 256 | 1 |
| Fragment 21 | menu | 50 | 22 | 46 | 2 |
| Fragment 22 | type\_of\_stay | 20 | 12 | 85 | 1 |
| Fragment 23 | near\_by\_sites | 50 | 14 | 73 | 1 |
| Fragment 24 | risk\_factor | 10 | 4 | 256 | 1 |

Considering the assumption we can calculate easily the size of 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 entire relation.

Having records on secondary storage, if you want to access them faster, then you need indexing. If a database is frequently queried and it is too large then it is supposed to have index to increase performance. There are various indexes used in databases. Here, we consider the following indexing scheme: Primary Index, Clustered Index and Secondary index. Based on the query, we decide what type of indexing file. The below table provides the details of fragment#, relation name, the attribute on which the index file is built and the type of index file built. For fragments from horizontal procedure, cluster index would be beneficial because most of the queries are based on name of the country and location. Fragment 19,20,22,23,24 are fit into one block. Therefore, using indexing is not really needed.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **fragment** | **relation** | **Indexing type** | **Indexing attribute(s)** | **Is a key?** |
| Fragment 1 | Site 1 | cluster | country | no |
| Fragment 2 | Site 2 | cluster | country | no |
| Fragment 3 | Site 3 | cluster | Country | no |
| Fragment 4 | Site 4 | cluster | country | no |
| Fragment 5 | Site 5 | cluster | Country | no |
| Fragment 6 | Site 6 | cluster | Country | no |
| Fragment 7 | Site 7 | cluster | Country | no |
| Fragment 8 | Site 8 | cluster | country | no |
| Fragment 9 | hotel 1 | cluster | location | no |
| Fragment 10 | hotel 2 | cluster | location | no |
| Fragment 11 | hotel 3 | cluster | location | no |
| Fragment 12 | hotel 4 | cluster | location | no |
| Fragment 13 | tourism\_agent | primary | Govt\_reg\_id | yes |
| Fragment 14 | Airlines | Primary | Flight\_no | yes |
| Fragment 15 | Schedule | primary | Schedule\_id | yes |
| Fragment 16 | Booking | primary | Book\_id | yes |
| Fragment 17 | Passenger | primary | Pass\_id | yes |
| Fragment 18 | transaction | primary | Trans\_id | yes |
| Fragment 19 | Nationality | NA | NA | no |
| Fragment 20 | percentage | NA | NA | no |
| Fragment 21 | menu | cluster | Schedule\_id | yes |
| Fragment 22 | type\_of\_stay | NA | NA | no |
| Fragment 23 | near\_by\_sites | NA | NA | no |
| Fragment 24 | risk\_factor | NA | NA | no |

In the first query, we use range to filter out records. Therefore, using B+ tree for first 8 fragments would benefit in access time. All other 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 particular record for all the relations.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Fragment** | **Relation** | **# of records** | **# no of data blocks** | **Index size per record** | **# of index records per block** | **# no of index blocks** | **# no of block access without indexing** | **# no of block access with indexing** |
| Fragment 1 | Site 1 | 300 | 15 | 4+10 | 73 | 5 | 15 | 3 |
| Fragment 2 | Site 2 | 200 | 10 | 4+10 | 73 | 3 | 10 | 3 |
| Fragment 3 | Site 3 | 400 | 19 | 4+10 | 73 | 6 | 19 | 4 |
| Fragment 4 | Site 4 | 200 | 10 | 4+10 | 73 | 3 | 10 | 3 |
| Fragment 5 | Site 5 | 150 | 8 | 4+10 | 73 | 3 | 8 | 3 |
| Fragment 6 | Site 6 | 200 | 10 | 4+10 | 73 | 3 | 10 | 3 |
| Fragment 7 | Site 7 | 100 | 5 | 4+10 | 73 | 2 | 5 | 2 |
| Fragment 8 | Site 8 | 300 | 15 | 4+10 | 73 | 5 | 15 | 4 |
| Fragment 9 | hotel 1 | 500 | 46 | 4+20 | 42 | 12 | 46 | 5 |
| Fragment 10 | hotel 2 | 700 | 64 | 4+20 | 42 | 17 | 64 | 6 |
| Fragment 11 | hotel 3 | 200 | 19 | 4+20 | 42 | 5 | 19 | 4 |
| Fragment 12 | hotel 4 | 400 | 37 | 4+20 | 42 | 10 | 37 | 5 |
| Fragment 13 | tourism\_agent | 200 | 8 | 4+4 | 124 | 2 | 8 | 2 |
| Fragment 14 | Airlines | 1000 | 50 | 4+4 | 124 | 9 | 50 | 4 |
| Fragment 15 | Schedule | 7000 | 69 | 4+4 | 124 | 55 | 69 | 7 |
| Fragment 16 | Booking | 1200 | 17 | 4+4 | 124 | 10 | 17 | 5 |
| Fragment 17 | Passenger | 5000 | 334 | 4+4 | 124 | 41 | 334 | 6 |
| Fragment 18 | transaction | 8000 | 223 | 4+4 | 124 | 65 | 223 | 8 |
| Fragment 19 | Nationality | 100 | 1 | NA | NA | NA | 1 | 1 |
| Fragment 20 | percentage | 200 | 1 | NA | NA | NA | 1 | 1 |
| Fragment 21 | menu | 50 | 2 | 4+4 | 124 | 1 | 2 | 1 |
| Fragment 22 | type\_of\_stay | 20 | 1 | NA | NA | NA | 1 | 1 |
| Fragment 23 | near\_by\_sites | 50 | 1 | NA | NA | NA | 1 | 1 |
| Fragment 24 | risk\_factor | 10 | 1 | NA | NA | NA | 1 | 1 |

Indexing the data file definitely reduces the number of block accesses needed to find a particular record from the data file. The complete statistics is showed in above table.

**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 + Latency + Block Transfer Time) \* 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 100kms from the other site and the speed of the transmission media connecting the sites is 10^7 meters/second. Propagation delay between the sites is computed as below.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Fragment** | **Relation** | **# of records** | **# no of data blocks** | **# no of block access with indexing** | **Local Query Time (ms)= (S+L+Tr)\* N** | **Remote Query Time(ms)** |
| Fragment 1 | Site 1 | 300 | 15 | 3 | 91.5 | 193.5 |
| Fragment 2 | Site 2 | 200 | 10 | 3 | 91.5 | 193.5 |
| Fragment 3 | Site 3 | 400 | 19 | 4 | 122 | 224 |
| Fragment 4 | Site 4 | 200 | 10 | 3 | 91.5 | 193.5 |
| Fragment 5 | Site 5 | 150 | 8 | 3 | 91.5 | 193.5 |
| Fragment 6 | Site 6 | 200 | 10 | 3 | 91.5 | 193.5 |
| Fragment 7 | Site 7 | 100 | 5 | 2 | 61 | 163 |
| Fragment 8 | Site 8 | 300 | 15 | 4 | 122 | 224 |
| Fragment 9 | hotel 1 | 500 | 46 | 5 | 152.5 | 254 |
| Fragment 10 | hotel 2 | 700 | 64 | 6 | 183 | 285 |
| Fragment 11 | hotel 3 | 200 | 19 | 4 | 122 | 224 |
| Fragment 12 | hotel 4 | 400 | 37 | 5 | 152.5 | 254 |
| Fragment 13 | tourism\_agent | 200 | 8 | 2 | 61 | 163 |
| Fragment 14 | Airlines | 1000 | 50 | 4 | 122 | 224 |
| Fragment 15 | Schedule | 7000 | 69 | 7 | 213.5 | 315.5 |
| Fragment 16 | Booking | 1200 | 17 | 5 | 152.5 | 254.5 |
| Fragment 17 | Passenger | 5000 | 334 | 6 | 183 | 285 |
| Fragment 18 | transaction | 8000 | 223 | 8 | 244 | 346 |
| Fragment 19 | Nationality | 100 | 1 | 1 | 30.5 | 132.5 |
| Fragment 20 | percentage | 200 | 1 | 1 | 30.5 | 132.5 |
| Fragment 21 | menu | 50 | 2 | 1 | 30.5 | 132.5 |
| Fragment 22 | type\_of\_stay | 20 | 1 | 1 | 30.5 | 132.5 |
| Fragment 23 | near\_by\_sites | 50 | 1 | 1 | 30.5 | 132.5 |
| Fragment 24 | risk\_factor | 10 | 1 | 1 | 30.5 | 132.5 |

1. **Propagation Delay** = (Distance between sites)/(Speed of Transmission media) = 100 \* 10^3 / 2 \* 10^6 which will be = 0.05s = 50 ms

Let us assume bandwidth of the network as 1MBps and data is exchanged between sites in form of packets. Package size is assumed to be 1500bytes. Transmission Time for a packet is given by,

1. **Packet Transmission Time** = (Size of packet) / Bandwidth = (1500 B) / (10^6 B/s) = 0.0015s, approximately equal to 2ms
2. **Remote Query Time** = Local Query Time + 2 \* Propagation Delay + Packet Transmission Time
3. **Remote Update Time** = Local Update Time + 2 \* Propagation Delay

Packet Transmission Time is included in Remote Query Time because; the result of the query will contain some data which is not negligible. But this data depends on the query, so it is assumed to be one packet, on an average. Using the formulations made above, the below table can be constructed.

**11. ALLOCATION AND REPLICATION:**

For allocating the fragments to sites we considered the **Redundant All Beneficial Sites method**. Transaction table is given below: consider there are four sites: S1, S2, S3, S4

| **Transaction** | **Originating sites** | **frequency** | **Fragment Access** |
| --- | --- | --- | --- |
| Q1 | S1 | 200 | F1 – 4 reads |
| S2 | 300 | F2 – 7 reads |
| S3 | 400 | F3 – 3 reads |
| S4 | 500 | F4 – 2 reads |
| Q2 | S1 | 250 | F5 – 8 reads |
| S2 | 150 | F6 – 4 reads |
| S3 | 50 | F7 – 6 reads |
| S4 | 100 | F8 – 2 reads |
| Q3 | S1 | 200 | F9 – 3 reads |
| S2 | 400 | F10 – 4 reads |
| S3 | 600 | F11 – 6 reads |
| S4 | 800 | F12 – 3 reads |
| Q4 | S1 | 900 | F13 – 8 reads |
| S2 | 300 | F14 – 7 reads |
| S3 | 200 | F15 – 3 reads |
| S4 | 100 | F16 – 25 reads |
| Q5 | S1 | 150 | F17 – 6 reads |
| S2 | 200 | F18 – 15 reads |
| S3 | 250 | F19 – 10 reads |
| S4 | 100 | F20 – 6 reads |
| Q6 | S1 | 250 | F21 – 7 reads |
| S2 | 350 | F22 – 5 reads |
| S3 | 600 | F23 – 7 reads |
| S4 | 300 | F24 – 9 reads |
| Q7 | S1 | 400 | F21 – 8 reads |
| S2 | 100 | F12 – 6 reads |
| S3 | 150 | F11 – 5 reads |
| S4 | 350 | F10 – 13 reads |
| Q8 | S1 | 300 | F1 – 15 reads |
| S2 | 200 | F2 – 9 reads |
| S3 | 500 | F8 – 4 reads |
| S4 | 300 | F7 – 3 reads |
| Q9 | S1 | 400 | F15 – 7 reads |
| S2 | 200 | F13 – 2 reads |
| S3 | 300 | F12 – 15 reads |
| S4 | 100 | F16 – 3 reads |
| Q10 | S1 | 700 | F23 – 7 reads |
| S2 | 900 | F2 – 2 reads |
| S3 | 300 | F5 – 6 reads |
| S4 | 200 | F22 – 49reads |

Since our sample queries are related only read (not update) will calculate only Benefit Computation (not Cost computation) of placing a fragment at a particular site. Let us proceed to Benefit Computation. Benefit computation is based on read queries. The benefit of placing each fragment at each site is given in the below table.

| **Fragments** | **Originating sites** | **Read query** | **# of reads \* Freq \* (Remote Time - Local Time)** | **Benefit(ms)** |
| --- | --- | --- | --- | --- |
| F1 | S1 | Q1, Q8 | (4\*200+15\*300) \* (193.5-91.5) | 540600 |
| S2 | none | none | 0 |
| S3 | none | none | 0 |
| S4 | none | none | 0 |
| F2 | S1 | none | none | 0 |
| S2 | Q10, Q8, Q1 | (2\*900+9\*200+7\*300)\*(193.5-91.5) | 581400 |
| S3 | none | none | 0 |
| S4 | none | none | 0 |
| F3 | S1 | none | none | 0 |
| S2 | none | none | 0 |
| S3 | Q1 | (3\*400)\*(224-122) | 122400 |
| S4 | none | none | 0 |
| F4 | S1 | none | none | 0 |
| S2 | none | none | 0 |
| S3 | none | none | 0 |
| S4 | Q1 | (2\*500 )\*(193.5-91.5) | 102000 |
| F5 | S1 | Q2 | (8\*250)\* (193.5-91.5) | 204000 |
| S2 | none | none | 0 |
| S3 | Q10 | (6\*300) \*(193.5-91.5) | 183600 |
| S4 | none | none | 0 |
| F6 | S1 | none | none | 0 |
| S2 | Q2 | (4\*150) \*(193.5-91.5) | 61200 |
| S3 | none | none | 0 |
| S4 | none | none | 0 |
| F7 | S1 | none | none | 0 |
| S2 | none | none | 0 |
| S3 | Q2 | (6\*50) \* (163-61) | 30600 |
| S4 | Q8 | (3\*300) \* (163-61) | 91800 |
| F8 | S1 | none | none | 0 |
| S2 | none | none | 0 |
| S3 | Q8 | (4\*500) \* (224-122) | 204000 |
| S4 | Q2 | (2\*100) \* (224-122) | 20400 |
| F9 | S1 | Q3 | (3\*200)\* (254-152.5) | 60900 |
| S2 | none | none | 0 |
| S3 | none | none | 0 |
| S4 | none | none | 0 |
| F10 | S1 | none | none | 0 |
| S2 | Q3 | (4\*400) \* (285-183) | 163200 |
| S3 | none | none | 0 |
| S4 | Q7 | (13\*350) \* (285-183) | 464100 |
| F11 | S1 | none | none | 0 |
| S2 | none | none | 0 |
| S3 | Q7, Q3 | (5\*150+6\*600) \* (224-122) | 443700 |
| S4 | none | none | 0 |
| F12 | S1 | none | none | 0 |
| S2 | Q7 | (6\*100) \* (254-152.5) | 60900 |
| S3 | Q9 | (15\*300) \* (254-152.5) | 456750 |
| S4 | Q3 | (3\*800) \* (254-152.5) | 243600 |
| F13 | S1 | Q4 | (8\*900)\* (163-61) | 734400 |
| S2 | Q9 | (2\*200) \* (163-61) | 40800 |
| S3 | none | none | 0 |
| S4 | none | none | 0 |
| F14 | S1 | none | none | 0 |
| S2 | Q4 | (7\*300)\* (224-122) | 214200 |
| S3 | none | none | 0 |
| S4 | none | none | 0 |
| F15 | S1 | Q9 | (7\*400)\* (315.5-213.5) | 285600 |
| S2 | none | none | 0 |
| S3 | Q4 | (3\*200)\* (315.5-213.5) | 61200 |
| S4 | none | none | 0 |
| F16 | S1 | none | none | 0 |
| S2 | none | none | 0 |
| S3 | none | none | 0 |
| S4 | Q4, Q9 | (25\*100+3\*100)\* (254.5-152.5) | 285600 |
| F17 | S1 | Q5 | (6\*150)\* (285-183) | 91800 |
| S2 | none | none | 0 |
| S3 | none | none | 0 |
| S4 | none | none | 0 |
| F18 | S1 | none | none | 0 |
| S2 | Q5 | (15\*200)\* (346-244) | 306000 |
| S3 | none | none | 0 |
| S4 | none | none | 0 |
| F19 | S1 | none | none | 0 |
| S2 | none | none | 0 |
| S3 | Q5 | (10\*250)\* (132.5-30.5) | 255000 |
| S4 | none | none | 0 |
| F20 | S1 | none | none | 0 |
| S2 | none | none | 0 |
| S3 | none | none | 0 |
| S4 | Q5 | (6\*100)\* (315.5-213.5) | 61200 |
| F21 | S1 | Q7, Q6 | (8\*400+7\*250)\* (315.5-213.5) | 504900 |
| S2 | none | none | 0 |
| S3 | none | none | 0 |
| S4 | none | none | 0 |
| F22 | S1 | none | none | 0 |
| S2 | Q6 | (5\*350) \* (315.5-213.5) | 178500 |
| S3 | none | none | 0 |
| S4 | Q10 | (30\*200)\* (315.5-213.5) | 612000 |
| F23 | S1 | Q10 | (7\*700)\* (315.5-213.5) | 499800 |
| S2 | none | none | 0 |
| S3 | Q6 | (7\*600)\* (315.5-213.5) | 428400 |
| S4 | none | none | 0 |
| F24 | S1 | none | none | 0 |
| S2 | none | none | 0 |
| S3 | none | none | 0 |
| S4 | Q6 | (9\*300)\* (315.5-213.5) | 275400 |

Since the cost of placing each fragment at each site is 0, the Benefit computation which is done it-self gives the Benefit – Cost value. So based on the Benefit table, we can infer the following:

Fragment1: is assigned at S1

Fragment2: is assigned at S2

Fragment3: is assigned at S3

Fragment4: is assigned at S4

Fragment5: is assigned at S3

Fragment6: is assigned at S2

Fragment7: is assigned at S4 and replicated to S3

Fragment8: is assigned at S3 and replicated to S4

Fragment9: is assigned at S1

Fragment10: is assigned at S4 and replicated to S3

Fragment11: is assigned at S3

Fragment12: is assigned at S3 and replicated to S2 and S4

Fragment13: is assigned at S1 and replicated to S2

Fragment14: is assigned at S2

Fragment15: is assigned at S1 and replicated to S3

Fragment16: is assigned at S4

Fragment17: is assigned at S1

Fragment18: is assigned at S2

Fragment19: is assigned at S3

Fragment20: is assigned at S4

Fragment21: is assigned at S1

Fragment22: is assigned at S4 and replicated to S2

Fragment23: is assigned at S1 and replicated to S3

Fragment24: is assigned at S4