NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA

Department of Information Technology

Advanced Database Systems

Assignment 1

Amazon Management System

Submitted by

Pankaj Kumar Magar - 222IT018
Vedant Parwal - 222IT034

Contents

Problem Description:

In this project, we aim to design a distributed database system integrating Amazon services like Amazon e-commerce, Amazon Prime, Amazon Pay, and Amazon courier services. Its main objective is to provide users with unified access to different services. Generally, if a user wants to access all the above services, then he/she needs to visit that particular service provider to get each service. In this case, each service provider will have policies that may differ significantly. We are providing a solution to integrate such services so that the same information is used at different sites. Some specific information about that service may change depending on the service provided.

Data Sources:

According to the problem statement, there are following data sources.

1. **Amazon e-commerce:** It is an e-commerce company that sells a wide variety of products. All products are tagged with price, rating, name, product id, available units, product color, in stock, and weight. Amazon requires users to register to buy a product by providing details such as their name, email address, password, and age. Additionally, customers can provide their contact information, such as their address, pincode and phone number. This information is used at different sites to access different services.
   Once registered, a customer can be both a buyer and a seller. Buyers can place orders by adding single or multiple items to the shopping cart. Once the order is placed, a user can view its order details which contain a unique order id, total price, delivery date, order date, order status, delivery address, quantity, shipping price and payment information. Shipping price varies depending on whether a buyer is a prime user or not. Similarly, a seller can sell its products using the same platform, which allows sellers to add seller-related information like its company name, description, average ratings, and website URL.

2. **Amazon couriers:** It offers secure and reliable delivery of consignments over multiple locations. Using this, sellers can ship their products, which includes carrier information such as carrier name, carrier email address, phone number, carrier ID, and rating. Amazon facility tracks each shipment using a unique shipment id, product name, delivery address, pick-up date, and delivery date.

3. **<u>Amazon Prime:</u>** It is a streaming service that offers various award-winning TV shows, movies, anime, documentaries, and more on thousands of internet-connected devices. Users can subscribe to a service by selecting a plan, and each plan has a unique subscription id, subscription type, subscription date, validity, and price. Media in each plan include a unique id, title, release date, and rating, and each media is tagged with a genre id,genre type, and genre description.

4. **<u>Amazon Pay</u>:** It lets users use the payment method already associated with their Amazon account to make payments. Besides paying for the services, users can also make payments to each other. Users must first register their account related details, such as account number, bank name, date of birth, and balance. Upon successful transaction completion, the transaction history is kept, containing the payment ID, the account number of the sender, the account number of the receiver, the date, and the amount of the transaction.

Overall, a single platform allows users to access multiple services. They can buy and sell products on Amazon e-commerce, ship products using Amazon Couriers, stream their favourite shows using Amazon Prime, and do transactions using Amazon Pay.

<u>Queries:</u>

Simple Queries :

1. List all products with prices under 5000.
2. Name of carrier service that has more than four ratings.
3. The title, media_rating of the television series with comedy as its genre.
4. Name of carrier service, rating that has less than or equal to four ratings.
5. Select product_id ,product_name whose in_stock unit is 0.
6. The title, media_rating of the television series with thriller as its genre

<u>Complex Queries :</u>

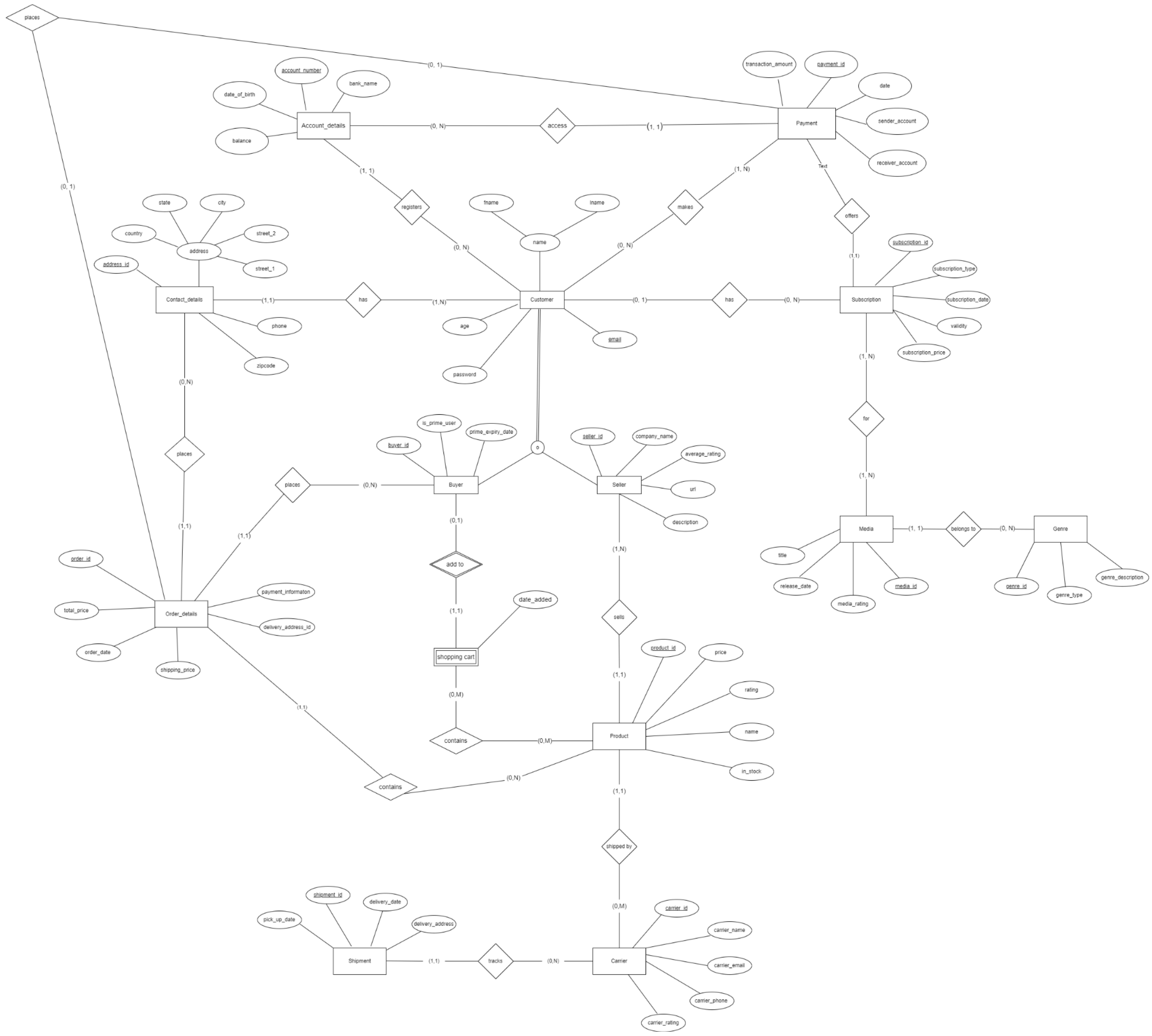
1. Fname, lname, email of customers that have not yet subscribed to amazon prime.
2. Amazon's prime subscription type, price which is subscribed to by most customers.

3. e-mail, age of customers whose age<18 and took the subscription_price >999.
4. Customers email whose prime subscription has expired.
5. Product name, price of the products in order_details which are having price more than current price.

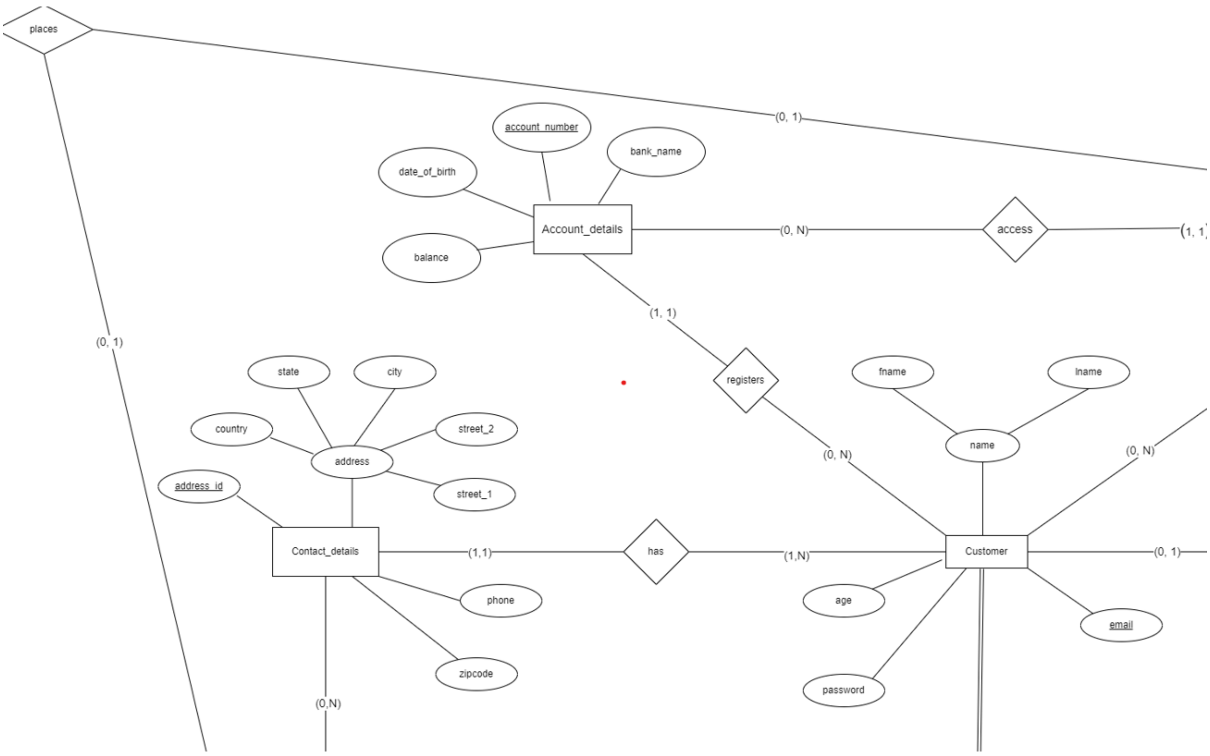
Constraints:

1. Products will only be added to the shopping cart if they are in stock.
2. If the quantity of an item in the shopping cart reaches zero, the item will be removed.
3. Media can only be streamed on a specified number of devices at a time.
4. In order to make a successful payment through Amazon pay, the user's wallet balance must be sufficient.
5. Courier Services will not accept a shipment if it does not provide delivery to the specified location.
6. Users whose age<18 cannot access the genre_type of “R rated”.
7. Only those sellers whose average rating >=2 will sell the products.
8. A user cannot do a transaction to himself using Amazon Pay.

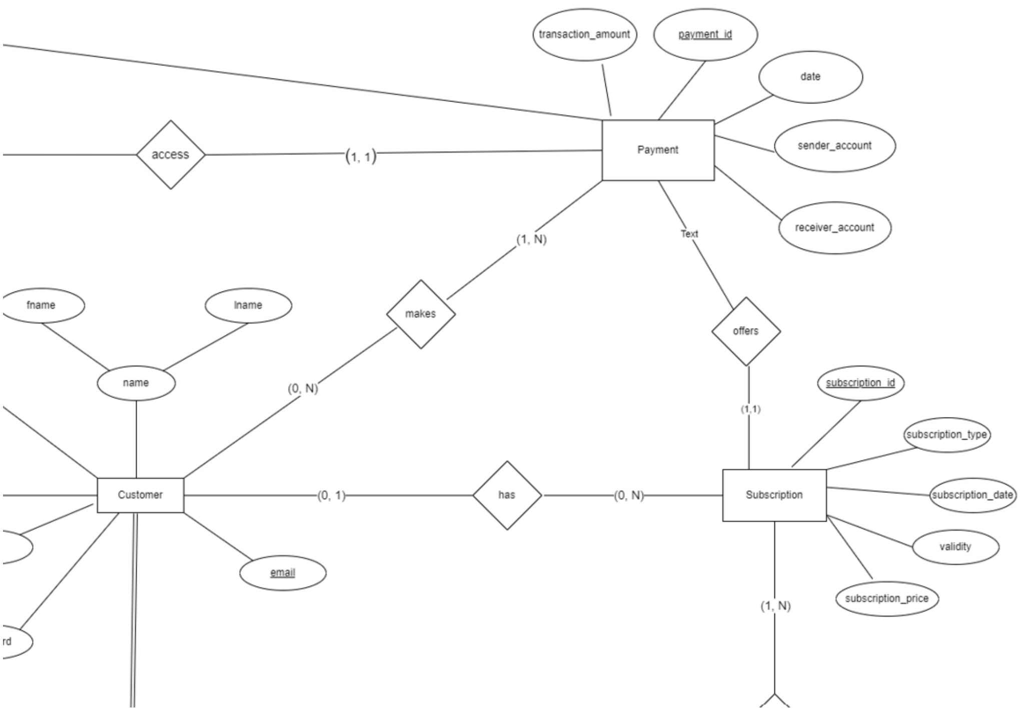
Entity Relationship Diagram :



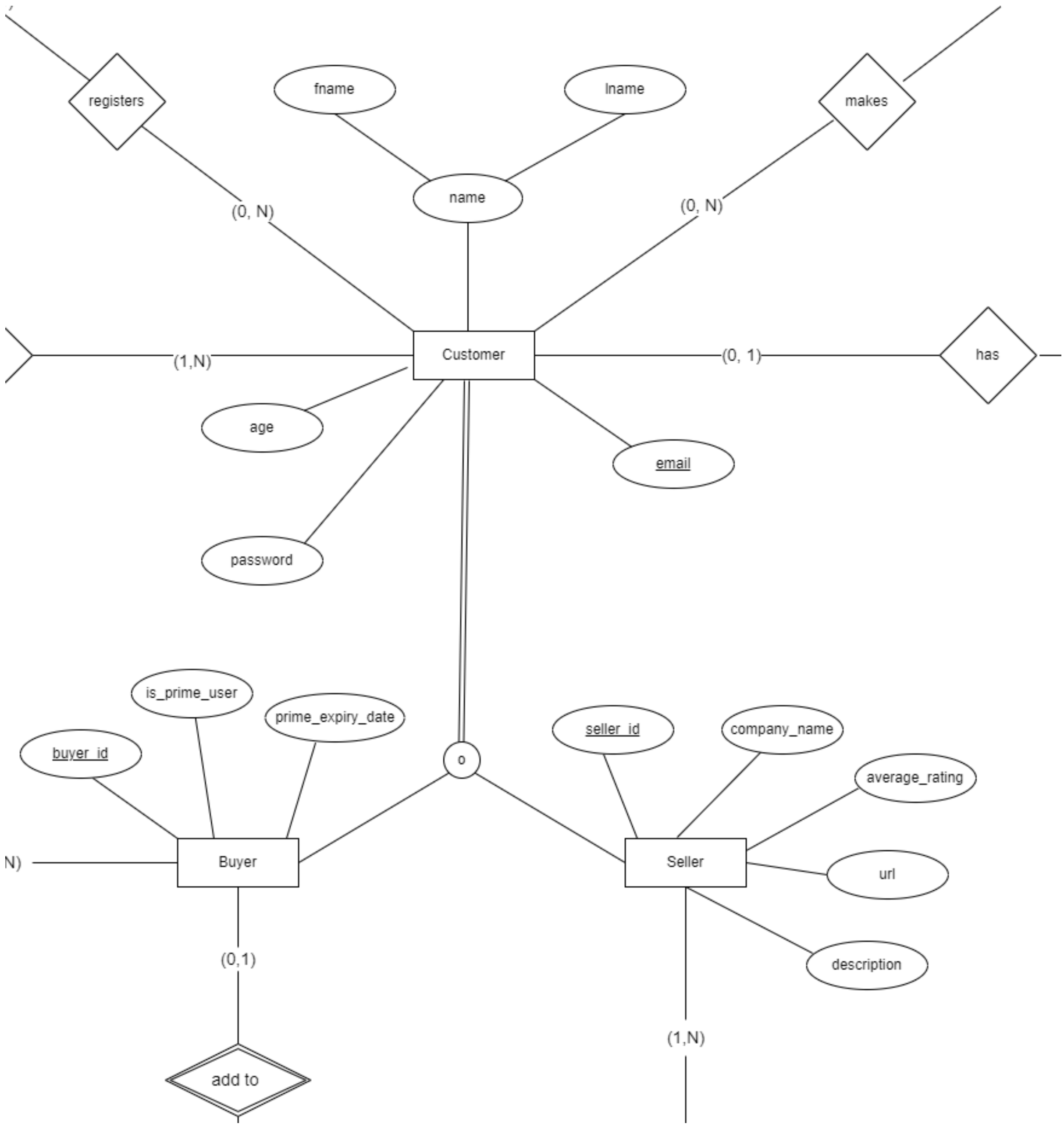
Top Left :-



Top Right :-

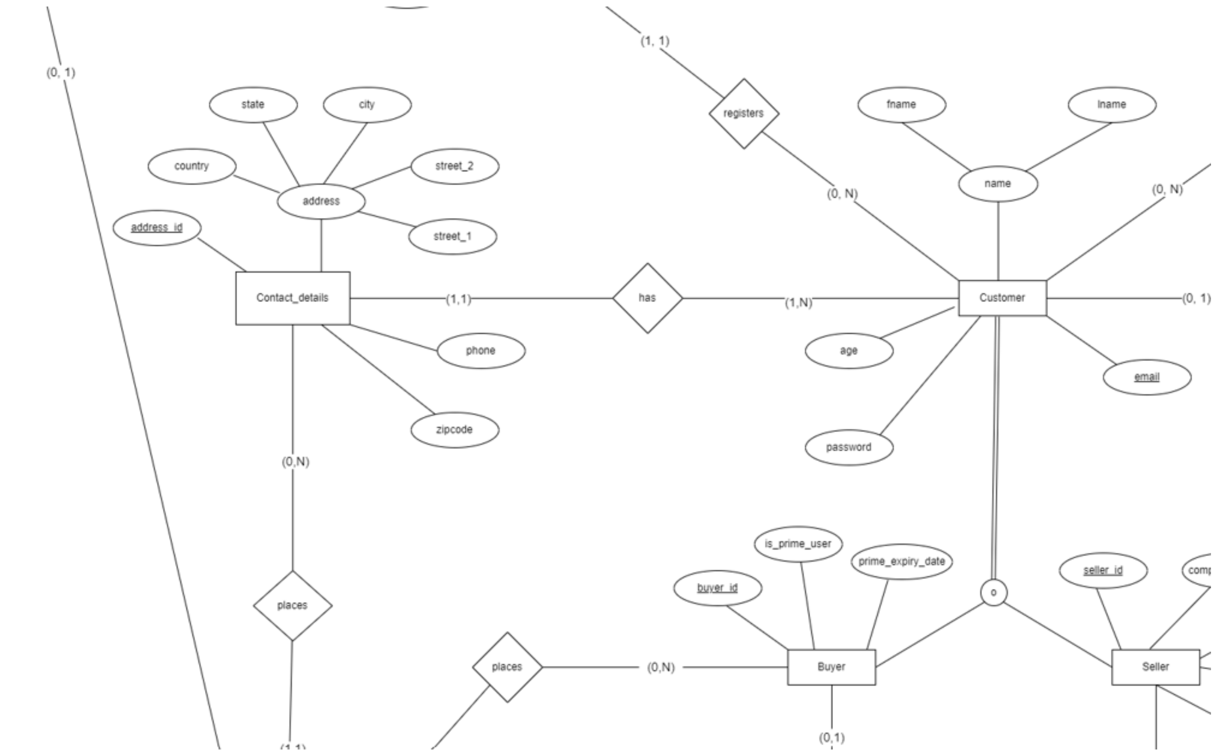


Middle : -

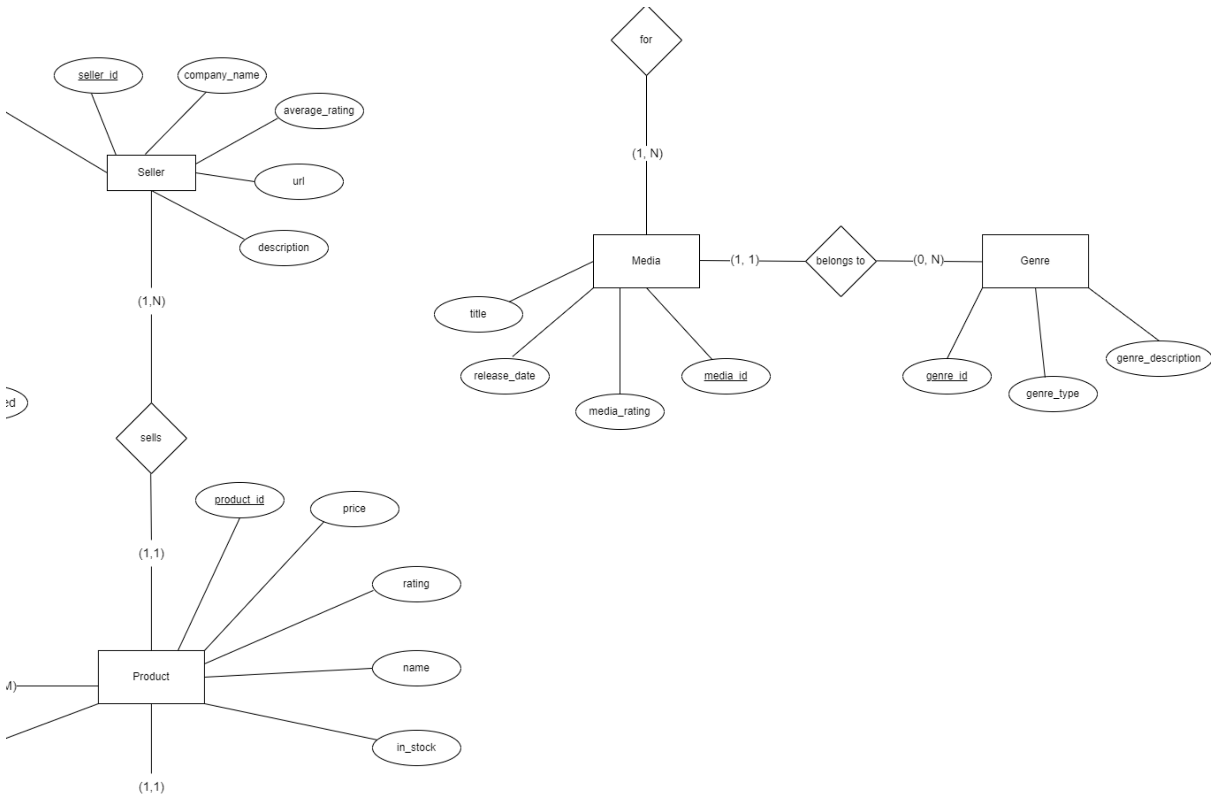


registers
fname
lname
makes
(0, N)
name
(0, N)
(1,N)
Customer
(0, 1)
has
age
email
password
is_prime_user
prime_expiry_date
seller_id
company_name
buyer_id
o
average_rating
N)
Buyer
Seller
url
(0,1)
description
(1,N)
add to

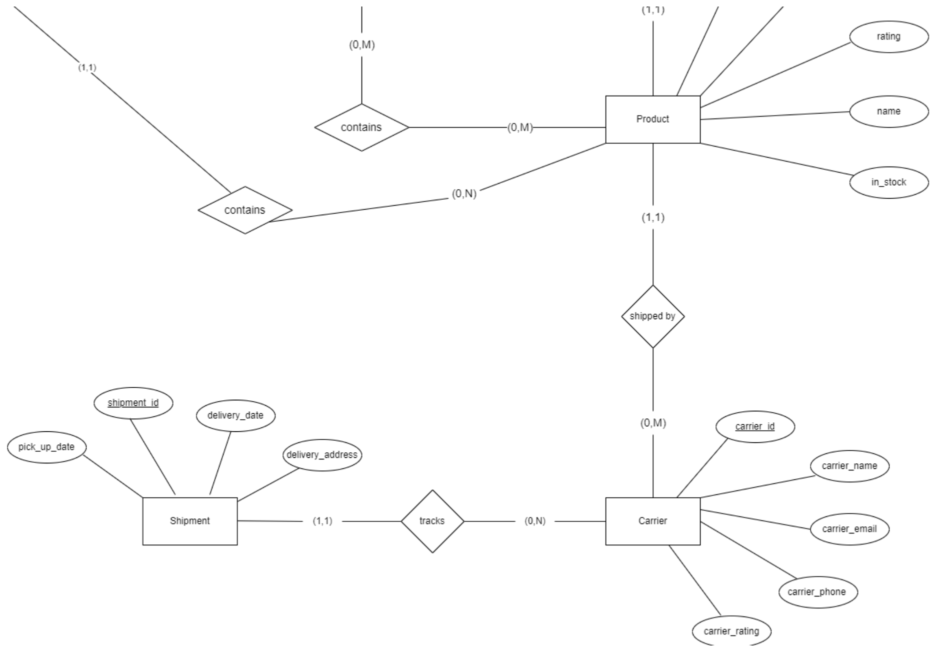
Middle Left : -



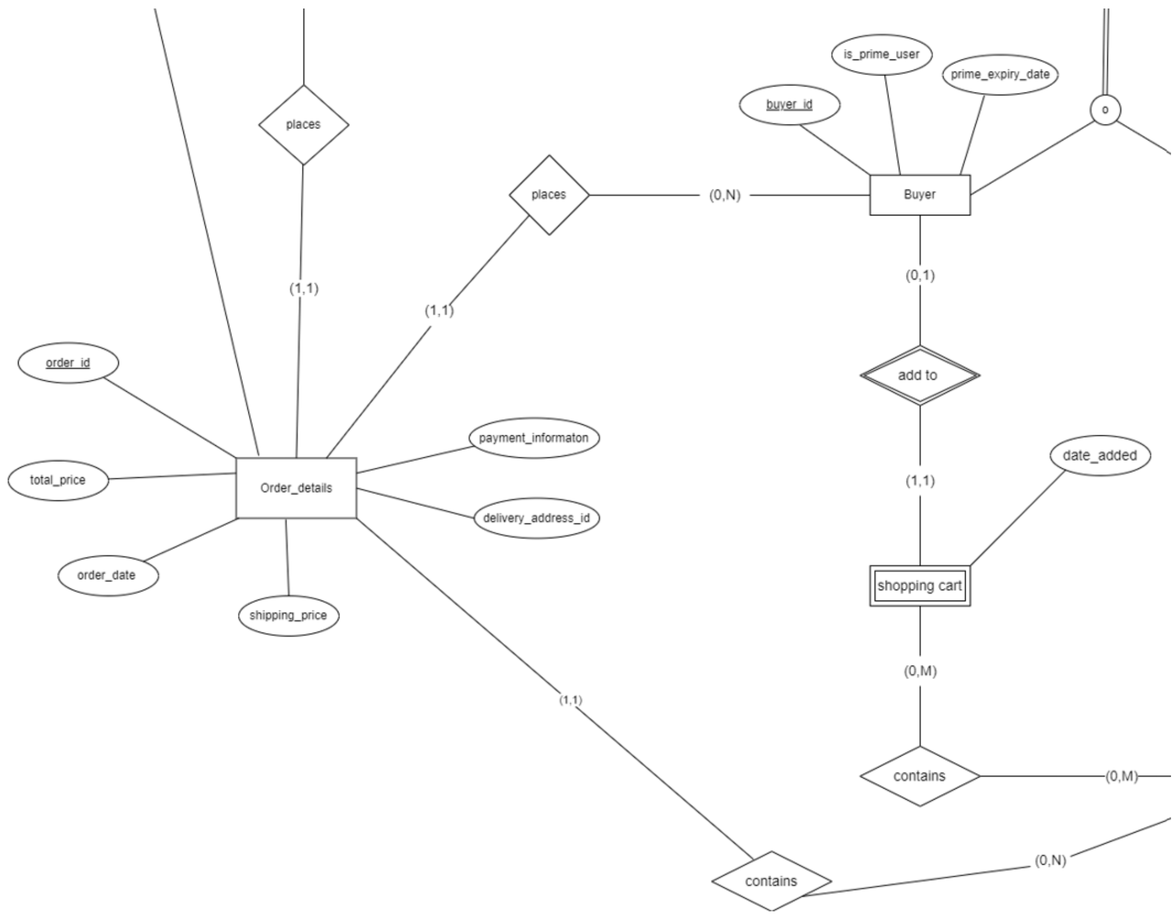
Middle Right : -



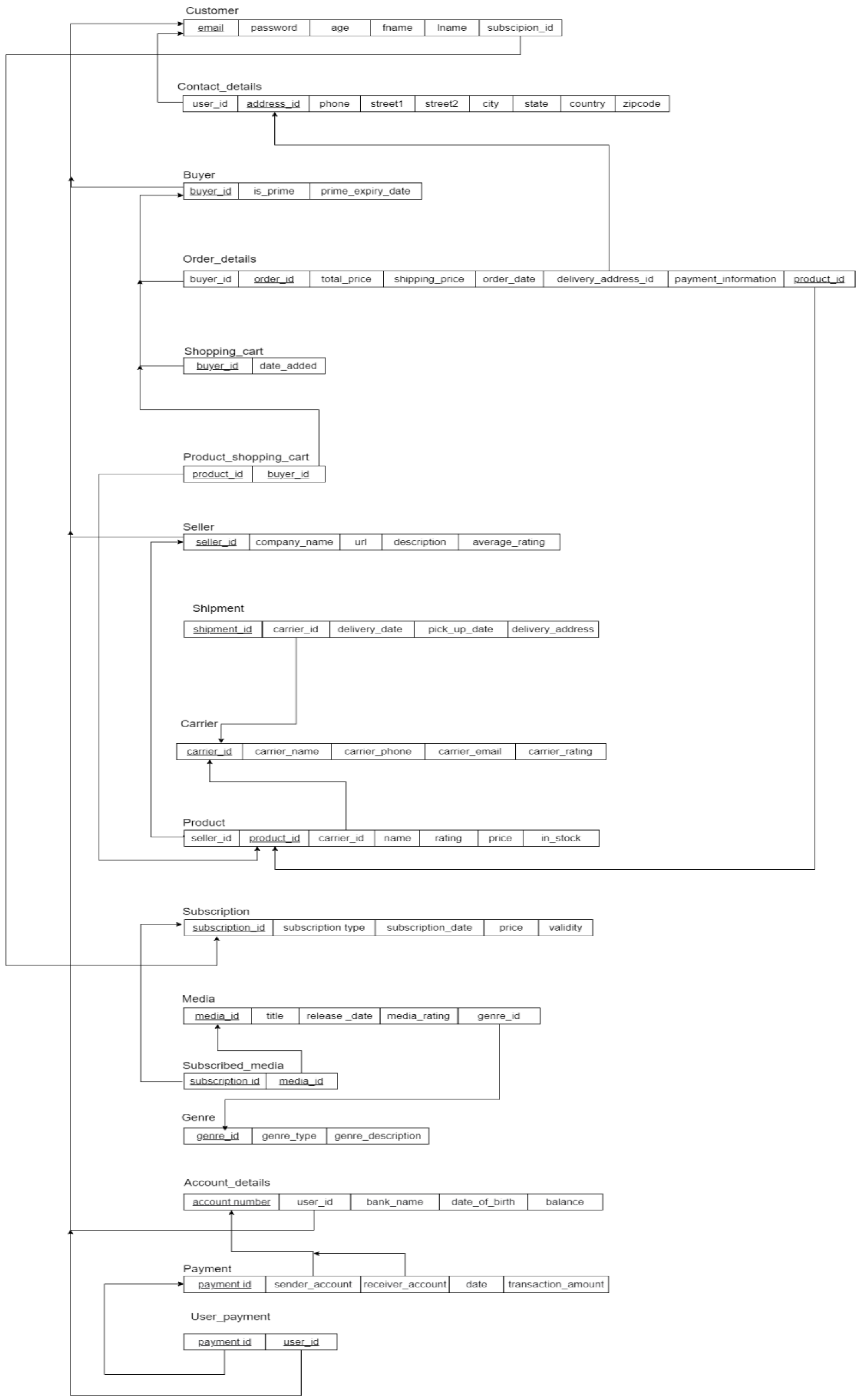
Bottom : -



Bottom Left : -



Relational Schema :



Normalization:

Normalization is the process of reorganizing data within a database so that users can utilize it for further queries and analysis. It is a database design technique that reduces data redundancy and eliminates undesirable characteristics like Insertion, Update and Deletion Anomalies. In addition to that it includes eliminating redundant and unstructured data and making the data appear similar across all records and fields.. Normalization rules divide larger tables into smaller tables and links them using relationships.

Different types of normal forms are used to eliminate or reduce redundancy in database tables.

1st Normal Form: A relation is in first normal form if every attribute in that relation is a single valued attribute. Conditions of 1st normal form:

- A relation will be 1NF if it contains an atomic value.
- No multi-valued attribute, composite attribute, and their combinations.

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 should be 1st 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.

Fragmentation:

The process of fragmentation involves breaking up the entire or complete database into numerous subtables or sub relations so that data can be stored in various systems. Fragments are the little parts of sub relations or subtables. These pieces, known as logical data units, are kept at multiple locations.

Advantages of Fragmentation:

1. Efficiency of database systems will increase as data is stored close to the usage site.
2. Provides better local processing.
3. Parallel execution can be done.
4. Query response time is improved.

Types of Fragmentation :

1. Horizontal Fragmentation.
    a. Primary Horizontal Fragmentation.
    b. Derived Horizontal Fragmentation.
2. Vertical Fragmentation.

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.

- **Disjointness:** Row or column must be present in at most one site, else will lead to inconsistent data.

Following are the lists of queries depicting the transactions in the Amazon Management System:

Query 1 : List all names, price of products with prices under 5000.

```
SELECT product_name, price FROM
Product as P where P.price < 5000
```

Query 2 : Name of carrier service, rating that has more than four ratings.

```
SELECT carrier_name, carrier_rating
FROM Carrier as C
WHERE C.carrier_rating > 4.
```

Query 3 : The title, media_rating of the television series with comedy as its genre.

```
SELECT title, media_rating
FROM Media as M, Genre as G
WHERE M.genre_id = G.genre_id
AND  G.genre_type = ‘comedy’
```

Query 4 : Customers email whose prime subscription has expired.

```
SELECT email
FROM Customer as C, Subscription as S
WHERE C. subscription_id = S.subscription_id
AND validity = ‘expired’.
```

Query 5 : fname, lname, email of customers that have not yet subscribed to amazon prime.

```
SELECT fname, lname, email
FROM Customer As C
WHERE C.subscription_id == NULL
```

Query 6 : The title, media_rating of the television series with thriller as its genre

```
SELECT title, media_rating
FROM Media as M, Genre as G
WHERE M.genre_id = G.genre_id
AND  genre_type = 'thriller'
```

Query 7 : Amazon's prime subscription type, price which is subscribed to by most customers.

```
SELECT subscription_type, subscription_price, count(*)
FROM Subscription
GROUP BY subscription_price, subscription_type
ORDER BY count(*) DESC
fetch first 1 row only;
```

Query 8 : Select product_id ,product_name whose in_stock unit is 0.

```
Select product_id, product_name from Product as P
where P.in_stock = 0.
```

Query 9 : e-mail, age of customers whose age<18 and took the subscription_price >999.

```
SELECT age, email FROM
Customer as C, Subscription as S
WHERE C. subscription_id = S.subscription_id
AND C.age<18 AND S.subscription_price > 999
```

Query 10 : Product name, price of the products in order_details which are having price more than current price.

```
SELECT product_name, product_price AS current_price
FROM  Product as P, Order_details as O
WHERE O.product_id = P.product_id
AND P.product_price < (O.total_price - O.shipping_price)
```

Query 11 : Name of carrier service, rating that has less than or equal to four ratings.

```
SELECT carrier_name, carrier_rating
FROM Carrier as C
WHERE C.carrier_rating <= 4.
```

Horizontal Fragmentation:

Horizontal fragmentation partitions the relation along its tuples of the relations. Every fragment will have the same number of attributes.

a. **Primary Horizontal Fragmentation:-** Fragmentation is based on the predicates defined ON THAT relation.

   Query2 : Name, rating, email of carrier service that has more than four ratings.
   Query11: Name of carrier service, rating that has less than or equal to four ratings.

   p1 = {carrier_rating > 4}
   p2 = {carrier_rating <= 4}
   carrier1 = $\sigma_{carier\_rating > 4}$ (carrier)
   carrier2 = $\sigma_{carier\_rating <= 4}$ (carrier)

   Here a query is accessing almost all attributes of carrier relation and predicate is defined over a range therefore we go for horizontal fragmentation.

b. **Derived Horizontal Fragmentation:-** Fragmentation is based on the predicates defined on OTHER relation.

   Query3 : The title,media_rating of the television series with comedy as its genre.

   Query6 : The title, media_rating of the television series with thriller as its genre.

p1: genre = “thriller”
p2: genre = comedy

genre1 = $\sigma_{\text{genre\_type = “thriller”}}$ (genre)

genre2 = $\sigma_{\text{genre\_type = “comedy”}}$ (genre)

media1 = media $\ltimes_{\text{genre\_id}}$ genre1
media2 = media $\ltimes_{\text{genre\_id}}$ genre2

Genre relation is fragmented based on genre_type. Primary key of genre is placed as foreign key in Media relation. Now if we fragment Media based on its attributes then for every insertion of genre_id in Media we need to check whether that genre_id is also present in Genre relation or not. To avoid this we fragment Media according to genre to place similar fragments from both relations together at a site and hence we apply derived horizontal fragmentation.

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 time and 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 split them 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 :

Assume frequency matrix as follows:

	S1	S2	S3	S4	Total Query
Q1	0	10	20	0	30
Q2	5	0	10	0	15
Q3	10	15	10	0	35
Q4	10	5	0	5	20
Q5	0	10	15	5	30
Q6	10	0	0	15	25
Q7	15	10	20	0	45
Q8	5	15	0	10	30
Q9	0	5	5	0	10
Q10	15	0	20	5	40
Q11	0	0	15	15	30

Relation-1: PRODUCT

- **Attribute Usage Matrix:**

	Product_id A1	Seller_id A2	Carrier_id A3	Name A4	Rating A5	Price A6	In_Stock A7
Q1	0	0	0	1	0	1	0
Q2	0	0	0	0	0	0	0
Q3	0	0	0	0	0	0	0
Q4	0	0	0	0	0	0	0

Q5	0	0	0	0	0	0	0
Q6	0	0	0	0	0	0	0
Q7	0	0	0	0	0	0	0
Q8	1	0	0	1	0	0	1
Q9	0	0	0	0	0	0	0
Q10	1	0	0	1	0	1	0
Q11	0	0	0	0	0	0	0

- **Attribute Affinity Matrix:**

	A1	A2	A3	A4	A5	A6	A7
A1	70	0	0	70	0	40	30
A2	0	0	0	0	0	0	0
A3	0	0	0	0	0	0	0
A4	70	0	0	100	0	70	30
A5	0	0	0	0	0	0	0
A6	40	0	0	70	0	70	0
A7	30	0	0	30	0	0	30

To calculate the Cluster Affinity matrix ordering we make use of the Bond Energy Algorithm (BEA). BEA groups the attributes which have more affinity and groups which have less affinity. For this we make use of the following functions.

Cont(Ai,Aj,Ak) = 2*[Bond(Ai,Aj) + Bond(Aj,Ak) - Bond(Ai,Ak)]

$$\text{Bond}(Ai,Aj) = \sum_{p=1}^{n} AA(Ap, Ai) * AA(Ap, Aj)$$

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)] : 0
  CONT(234): 2[Bond(23)+Bond(34)-Bond(24)] : 0

The value of CONT(031) = CONT(132) = CONT(234) = 0, so the order of this attribute need not be changed, the order is T(123).

- Attribute A4:
  CONT(041): 2[Bond(04)+Bond(41)-Bond(01)] : 31200
  CONT(142): 2[Bond(14)+Bond(42)-Bond(12)] : 31200
  CONT(243): 2[Bond(24)+Bond(43)-Bond(23)] : 0
  CONT(345): 2[Bond(34)+Bond(45)-Bond(35)] : 0

The value of CONT(041) is larger.So the order will be T(4123).

- Attribute A5:
  CONT(054)=2[Bond(05)+Bond(54)-Bond(04)] : 0
  CONT(451)=2[Bond(45)+Bond(51)-Bond(41)] : -31200
  CONT(152)=2[Bond(15)+Bond(52)-Bond(12)] : 0
  CONT(253)=2[Bond(25)+Bond(53)-Bond(23)] : 0
  CONT(356)=2[Bond(35)+Bond(56)-Bond(36)] : 0

The value of CONT(054) is larger.So the order will be T(54123).

- Attribute A6:
  CONT(065)=2[Bond(06)+Bond(65)-Bond(05)] : 0
  CONT(564)=2[Bond(56)+Bond(64)-Bond(54)] : 29400
  CONT(461)=2[Bond(46)+Bond(61)-Bond(41)] : 19200
  CONT(162)=2[Bond(16)+Bond(62)-Bond(12)] : 21000
  CONT(263)=2[Bond(26)+Bond(63)-Bond(23)] : 0
  CONT(367)=2[Bond(36)+Bond(67)-Bond(37)] : 6600

The value of CONT(564) is larger. So the order will be T(564123).

- Attribute A7:
  CONT(075)=2[Bond(07)+Bond(75)-Bond(05)] : 0
  CONT(576)=2[Bond(57)+Bond(76)-Bond(56)] : 6600
  CONT(674)=2[Bond(67)+Bond(74)-Bond(64)] : -10800
  CONT(471)=2[Bond(47)+Bond(71)-Bond(41)] : -9000
  CONT(172)=2[Bond(17)+Bond(72)-Bond(12)] : 10200
  CONT(273)=2[Bond(27)+Bond(73)-Bond(23)] : 0
  CONT(378)=2[Bond(37)+Bond(78)-Bond(38)] : 0

The value of CONT(172) is larger. So the order will be T(5641723).

So, overall the final order of the attributes in the relation is T(5641723).

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

Cluster Affinity Matrix:

	A5	**A6**	**A4**	**A1**	**A7**	**A2**	**A3**
A5	0	0	0	0	0	0	0
A6	0	70	70	40	0	0	0
A4	0	70	100	70	30	0	0
A1	0	40	70	70	30	0	0
A7	0	0	30	30	30	0	0
A2	0	0	0	0	0	0	0
A3	0	0	0	0	0	0	0

Maximise the total access to EACH segment
Minimise the total access to BOTH segment
TA – Set of attributes in fragment f1
TB – Set of attributes in fragment f2
TQ – Number of applications accesses only TA
BQ – Number of applications accesses only TB

OQ – Number of applications accesses both TA and TB
CTQ – Total number of access to attributes by applications that access only TA
CBQ – Total number of access to attributes by applications that access only TB
COQ – Total number of access to attributes by applications that access both TA and TB

Z = CTQ x CBQ - COQ*COQ

Now, we'll select the maximum Z value.

Partition Algorithm:

1. **TA= {A5}** **TB= {A6, A4, A1, A7, A2, A3}**
   TQ= { }
   BQ= { Q1, Q8, Q10 }
   OQ= { }
   CTQ=0
   CBQ=30+30+40=100
   COQ=0

   Z = CTQ x CBQ - COQ*COQ
   Z = 0 x 100 - 0
   Z = 0

2. **TA= {A5,A6}** **TB= {A4, A1, A7, A2, A3}**
   TQ= { }
   BQ= {Q8}
   OQ= { Q1,Q10 }
   CTQ=0
   CBQ=30
   COQ=70

   Z = -4900

3. **TA= {A5,A6,A4}** **TB= {A1, A7, A2, A3}**
   TQ= {Q1}
   BQ= { }
   OQ= { Q8,Q10 }
   CTQ=30
   CBQ=0
   COQ=70

   Z = -4900

4. **TA= {A5,A6,A4,A1}** **TB= { A7, A2, A3}**
   TQ= {Q1,Q10} CTQ=70
   BQ= { } CBQ=0
   OQ= { Q8} COQ=30

   Z = -900

5. **TA= {A5,A6,A4,A1,A7}** **TB= {A2, A3}**
   TQ= {Q1,Q8,Q10} CTQ=100
   BQ= { } CBQ=0
   OQ= { } COQ=0

   Z = 0

6. **TA= {A5,A6,A4,A1,A7,A2}** **TB= {A3}**
   TQ= {Q1,Q8,Q10} CTQ=100
   BQ= { } CBQ=0
   OQ= { } COQ=0

   Z = 0

Relation - 2 : SUBSCRIPTION

- **Attribute Usage Matrix:**

	Subscription_id A1	Subscription_type A2	Subscription_date A3	Subscription_price A4	Validity A5
Q1	0	0	0	0	0
Q2	0	0	0	0	0
Q3	0	0	0	0	0
Q4	1	0	0	0	1
Q5	0	0	0	0	0
Q6	0	0	0	0	0
Q7	0	1	0	1	0
Q8	0	0	0	0	0
Q9	1	0	0	1	0
Q10	0	0	0	0	0
Q11	0	0	0	0	0

- **Attribute Affinity Matrix:**

	A1	A2	A3	A4	A5
A1	30	0	0	10	20
A2	0	45	0	45	0
A3	0	0	0	0	0
A4	10	45	0	55	0
A5	20	0	0	0	20

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)] : -900
  CONT(234): 2[Bond(23)+Bond(34)-Bond(24)] : -9000

The value of CONT(031) = CONT(132) = CONT(234) = 0, so the order of this attribute need not be changed, the order is T(123).

- Attribute A4:
  CONT(041): 2[Bond(04)+Bond(41)-Bond(01)] : 1700
  CONT(142): 2[Bond(14)+Bond(42)-Bond(12)] : 9800
  CONT(243): 2[Bond(24)+Bond(43)-Bond(23)] : 9000
  CONT(345): 2[Bond(34)+Bond(45)-Bond(35)] : 400

The value of CONT(142) is larger.So the order will be T(1423).

- Attribute A5:
  CONT(051)=2[Bond(05)+Bond(51)-Bond(01)] : 2000
  CONT(154)=2[Bond(15)+Bond(54)-Bond(14)] : 700
  CONT(452)=2[Bond(45)+Bond(52)-Bond(42)] :-8600
  CONT(253)=2[Bond(25)+Bond(53)-Bond(23)] : 0
  CONT(356)=2[Bond(35)+Bond(56)-Bond(36)] : 0

The value of CONT(051) is larger.So the order will be T(51423).

So, overall the final order of the attributes in the relation is T(51423).

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

Cluster Affinity Matrix:

	A5	**A1**	**A4**	**A2**	**A3**
A5	20	20	0	0	0
A1	20	30	10	0	0
A4	0	10	55	45	0
A2	0	0	45	45	0
A3	0	0	0	0	0

- **Partition Algorithm:**

1. **TA= {A5}**
   TQ= { }
   BQ= { Q7, Q9 }
   OQ= { Q4 }

   TB= {A1, A4, A2, A3}
   CTQ=0
   CBQ=45+10=55
   COQ= 20

   Z = CTQ x CBQ - COQ*COQ

   Z = -400

2. **TA= {A5,A1}**
   TQ= {Q4 }
   BQ= { Q7}
   OQ= { Q9 }

   TB= {A4, A2, A3}
   CTQ=20
   CBQ=45
   COQ= 10

   Z = 800

3. **TA= {A5,A1,A4}**
   TQ= {Q4,Q9 }
   BQ= { }
   OQ= { Q7 }

   TB= { A2, A3}
   CTQ = 30
   CBQ = 0
   COQ = 45

   Z = -2025

4. **TA= {A5,A1,A4,A2}**
   TQ= {Q4,Q7,Q9 }
   BQ= { }
   OQ= { }

   Z = 0

TB= { A3}
CTQ = 75
CBQ = 0
COQ = 0

Therefore, **MAX(Z) = 800.**

Based on the above procedure of vertical fragmentation on the SUBSCRIPTION relation, we will vertically fragment the SUBSCRIPTION relation:

SUBSCRIPTION 1 (15)
SUBSCRIPTION 2 (234).

Relation - 3 : CUSTOMER

- **Attribute Usage Matrix:**

	email A1	password A2	age A3	fname A4	lname A5	subscription_id A6
Q1	0	0	0	0	0	0
Q2	0	0	0	0	0	0
Q3	0	0	0	0	0	0
Q4	1	0	0	0	0	1
Q5	1	0	0	1	1	1
Q6	0	0	0	0	0	0
Q7	0	0	0	0	0	0
Q8	0	0	0	0	0	0
Q9	1	0	1	0	0	1
Q10	0	0	0	0	0	0

Q11	0	0	0	0	0	0

- **Attribute Affinity Matrix:**

	A1	**A2**	**A3**	**A4**	**A5**	**A6**
A1	60	0	10	30	30	60
A2	0	0	0	0	0	0
A3	10	0	10	0	0	10
A4	30	0	0	30	30	30
A5	30	0	0	30	30	30
A6	60	0	10	30	30	60

Ordering of the attributes:

- Attribute A3:
  CONT(031): 2[Bond(03)+Bond(31)-Bond(01)] : 2600
  CONT(132): 2[Bond(13)+Bond(32)-Bond(12)] : 2600
  CONT(234): 2[Bond(23)+Bond(34)-Bond(24)] : 0

The value of CONT(031) = CONT(132) = CONT(234) = 0, so the order of this attribute need not be changed, the order is T(123).

- Attribute A4:
  CONT(043): 2[Bond(04)+Bond(43)-Bond(03)] : 1200
  CONT(341): 2[Bond(34)+Bond(41)-Bond(31)] : 9400
  CONT(142): 2[Bond(14)+Bond(42)-Bond(12)] : 10800
  CONT(245): 2[Bond(24)+Bond(45)-Bond(25)] : 0

The value of CONT(041) is larger.So the order will be T(4123).

- Attribute A5:
  CONT(053)=2[Bond(05)+Bond(53)-Bond(03)] : 1200
  CONT(351)=2[Bond(35)+Bond(51)-Bond(31)] : 9400

CONT(154)=2[Bond(15)+Bond(54)-Bond(14)] : 7200
CONT(452)=2[Bond(45)+Bond(52)-Bond(42)] : 7200
CONT(256)=2[Bond(25)+Bond(56)-Bond(26)] : 0

- Attribute A6:
  CONT(063)=2[Bond(06)+Bond(63)-Bond(03)] : 2600
  CONT(365)=2[Bond(36)+Bond(65)-Bond(35)] : 12200
  CONT(561)=2[Bond(56)+Bond(61)-Bond(51)] : 18200
  CONT(164)=2[Bond(16)+Bond(64)-Bond(14)] : 18200
  CONT(267)=2[Bond(26)+Bond(67)-Bond(27)] : 0

So, overall the final order of the attributes in the relation is T(35142).

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

Cluster Affinity Matrix:

	A3	**A5**	**A6**	**A1**	**A4**	**A2**
A3	10	0	10	10	0	0
A5	0	30	30	30	30	0
A6	10	30	60	60	30	0
A1	10	30	60	60	30	0
A4	0	30	30	30	30	0
A2	0	0	0	0	0	0

- **Partition Algorithm:**

1. **TA= {A3}** **TB= {A5, A6, A1, A4, A2}**
   TQ= { } CTQ = 0
   BQ= { Q4, Q5 } CBQ = 20+30= 50
   OQ= {Q9} COQ = 10

   Z = CTQ x CBQ - COQ*COQ

Z = -100

2. **TA= {A3,A5}** **TB= {A6, A1, A4, A2}**
   TQ= { } CTQ = 0
   BQ= {Q4} CBQ = 20
   OQ= { Q5, Q9 } COQ = 30 + 10 =40

   Z = -1600

3. **TA= {A3,A5,A6}** **TB= {A1, A4, A2}**
   TQ= { } CTQ= 0
   BQ= { } CBQ= 0
   OQ= { Q4,Q5, Q9} COQ = 20+30+10 = 60

   Z = -3600

4. **TA= {A3,A5,A6,A1}** **TB= {A4, A2}**
   TQ= {Q4, Q9} CTQ = 20+10 = 30
   BQ= { } CBQ = 0
   OQ= {Q5} COQ = 30

   Z = -900

5. **TA= {A3,A5,A6,A1, A4}** **TB= {A2}**
   TQ= {Q4, Q5, Q9} CTQ = 20+30+10 = 60
   BQ = { } CBQ = 0
   OQ = { } COQ = 0

   Z = 0

Most of the values of Z are 0 or negative so fragmentation is application dependent. so we don’t want to do any fragmentation here.

Relation - 4 :ORDER_DETAILS

- **Attribute Usage Matrix:**

	buyer_id A1	order_id A2	total_price A3	shipping_price A4	order_date A5	delivery_address_id A6	payment_information A7	product_id 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	0	0	0	0	0	0	0	0
Q6	0	1	0	0	0	0	0	0
Q7	0	0	0	0	0	0	0	0
Q8	0	0	0	0	0	0	0	0
Q9	0	0	0	0	0	0	0	0
Q10	0	0	1	1	0	0	0	1
Q11	0	0	0	0	0	0	0	0

- **Attribute Affinity Matrix:**

	A1	**A2**	**A3**	**A4**	**A5**	**A6**	**A7**	**A8**
A1	0	0	0	0	0	0	0	0
A2	0	25	0	0	0	0	0	0
A3	0	0	40	40	0	0	0	40

A4	0	0	40	40	0	0	0	40
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	40	40	0	0	0	40

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)] : 0
  CONT(234): 2[Bond(23)+Bond(34)-Bond(24)] : 9600

The value of CONT(234) is larger.So the order will be T(123).

- Attribute A4:
  CONT(041): 2[Bond(04)+Bond(41)-Bond(01)] : 0
  CONT(142): 2[Bond(14)+Bond(42)-Bond(12)] : 0
  CONT(243): 2[Bond(24)+Bond(43)-Bond(23)] : 9600
  CONT(345): 2[Bond(34)+Bond(45)-Bond(35)] : 9600

The value of CONT(243) is larger.So the order will be T(1243).

- Attribute A5:
  CONT(051): 2[Bond(05)+Bond(51)-Bond(01)] : 0
  CONT(152): 2[Bond(15)+Bond(52)-Bond(12)] : 0
  CONT(254): 2[Bond(25)+Bond(54)-Bond(24)] : 0
  CONT(453): 2[Bond(45)+Bond(53)-Bond(43)] : -9600
  CONT(356): 2[Bond(35)+Bond(56)-Bond(36)] : 0

The value of CONT(051) is larger.So the order will be T(51243).

- Attribute A6:
  CONT(065)=2[Bond(06)+Bond(65)-Bond(05)] : 0
  CONT(561)=2[Bond(56)+Bond(61)-Bond(51)] : 0
  CONT(162)=2[Bond(16)+Bond(62)-Bond(12)] : 0
  CONT(264)=2[Bond(26)+Bond(64)-Bond(24)] : 0

CONT(463)=2[Bond(46)+Bond(63)-Bond(43)] : -9600
CONT(367)=2[Bond(36)+Bond(67)-Bond(37)] : 0

The value of CONT(065) is larger.So the order will be T(651243).

- Attribute A7:
  CONT(076)=2[Bond(07)+Bond(76)-Bond(06)] : 0
  CONT(675)=2[Bond(67)+Bond(75)-Bond(65)] : 0
  CONT(571)=2[Bond(57)+Bond(71)-Bond(51)] : 0
  CONT(172)=2[Bond(17)+Bond(72)-Bond(12)] : 0
  CONT(274)=2[Bond(27)+Bond(74)-Bond(24)] : 0
  CONT(473)=2[Bond(47)+Bond(73)-Bond(43)] : -9600
  CONT(378)=2[Bond(37)+Bond(78)-Bond(38)] : -9600

The value of CONT(076) is larger.So the order will be T(7651243).

- Attribute A8:

  CONT(087)=2[Bond(08)+Bond(87)-Bond(07)] : 0
  CONT(786)=2[Bond(78)+Bond(86)-Bond(76)] : 0
  CONT(685)=2[Bond(68)+Bond(85)-Bond(65)] : 0
  CONT(581)=2[Bond(58)+Bond(81)-Bond(51)] : 0
  CONT(182)=2[Bond(18)+Bond(82)-Bond(12)] : 0
  CONT(284)=2[Bond(28)+Bond(84)-Bond(24) : 9600
  CONT(483)=2[Bond(48)+Bond(83)-Bond(43)] : 9600
  CONT(389)=2[Bond(38)+Bond(89)-Bond(39)] : 9600

The value of CONT(284) is larger.So the order will be T(76512843).

So, overall the final order of the attributes in the relation is T(76512843).

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

Cluster Affinity Matrix:

	A7	A6	A5	A1	A2	A8	A4	A3
A7	0	0	0	0	0	0	0	0
A6	0	0	0	0	0	0	0	0

A5	0	0	0	0	0	0	0	0
A1	0	0	0	0	0	0	0	0
A2	0	0	0	0	25	0	0	0
A8	0	0	0	0	0	40	40	40
A4	0	0	0	0	0	40	40	40
A3	0	0	0	0	0	40	40	40

- **Partition Algorithm:**

1. **TA= {A7}**
   TQ= { }
   BQ= { Q6, Q10 }
   OQ= { }

   TB= {A6, A5, A1, A2, A8, A4, A3}
   CTQ=0
   CBQ=25+40=65
   COQ=0

   Z = CTQ x CBQ - COQ*COQ

   Z = 0

2. **TA= {A7,A6}**
   TQ= { }
   BQ= { Q6, Q10 }
   OQ= { }

   TB= {A5, A1, A2, A8, A4, A3}
   CTQ=0
   CBQ=25+40=65
   COQ=0

   Z = 0

3. **TA= {A7,A6,A5}**
   TQ= { }
   BQ= { Q6, Q10 }
   OQ= { }

   TB= {A1, A2, A8, A4, A3}
   CTQ=0
   CBQ=25+40=65
   COQ=0

   Z = 0

4. **TA= {A7,A6,A5,A1}**
   TQ= { }
   BQ= { Q6, Q10 }

   TB= {A2, A8, A4, A3}
   CTQ=0
   CBQ=25+40=65

OQ= { }	COQ=0

Z = 0

5. **TA= {A7,A6,A5,A1,A2}** **TB= {A8, A4, A3}**
   TQ= {Q6 } CTQ=25
   BQ= { Q10 } CBQ=40
   OQ= { } COQ=0

   Z = 1000

6. **TA= {A7,A6,A5,A1,A2,A8}** **TB= {A4, A3}**
   TQ= {Q6 } CTQ=25
   BQ= { } CBQ=0
   OQ= { Q10 } COQ=40

   Z = -1600

7. **TA= {A7,A6,A5,A1,A2,A8,A4}** **TB= {A3}**
   TQ= {Q6 } CTQ=25
   BQ= { } CBQ=0
   OQ= { Q10 } COQ=40

   Z = -1600

Based on the above procedure of vertical fragmentation on the ORDER_DETAILS relation, we will vertically fragment the SUBSCRIPTION relation:

ORDER_DETAILS 1 (76512)
ORDER_DETAILS 2 (843).

Total fragments we have found till now are as follows:

Fragment	**Relations**
Fragment 1	CARRIER 1
Fragment 2	CARRIER 2
Fragment 3	GENRE 1
Fragment 4	GENRE 2
Fragment 5	MEDIA 1
Fragment 6	MEDIA 2
Fragment 7	PRODUCT
Fragment 8	SUBSCRIPTION 1
Fragment 9	SUBSCRIPTION 2
Fragment 10	CUSTOMER
Fragment 11	ORDER_DETAILS 1
Fragment 12	ORDER_DETAILS 2

Data Allocation and 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 Read	Remote Read	(Remote - Local)
Fragment 1	100	300	200
Fragment 2	100	300	200
Fragment 3	100	300	200
Fragment 4	150	250	100
Fragment 5	400	700	300
Fragment 6	300	400	100
Fragment 7	400	700	300
Fragment 8	400	750	350
Fragment 9	200	350	150
Fragment 10	150	270	120
Fragment 11	300	150	150
Fragment 12	200	100	100

Transaction	Originating Sites	Frequency	Fragment access
Q1	S2,S3	30	F7 - 2 Read
Q2	S1,S3	15	F1 - 1 Read
Q3	S1,S2,S3	35	F4 - 2 Read F6 - 1 Read
Q4	S1,S2,S4	20	F10 -2 Read F8 - 2 Read
Q5	S2,S3,S4	30	F10 - 4 Read
Q6	S1,S4	25	F3 - 2 Read F5 - 1 Read
Q7	S1,S2,S3	45	F9 - 2 Read
Q8	S1,S2,S4	30	F7 - 3 Read
Q9	S2,S3	10	F9 - 3 Read F10 - 2 Read
Q10	S1,S3,S4	40	F7 - 3 Read F12 - 3 Read
Q11	S3, S4	30	F2 - 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**

Fragment	Site	Query	Benefit	Benefit - Cost
F1	S1	Q2	(1*15)*200	3000
	S2	-	-	0
	S3	Q2	(1*15)*200	3000
	S4	-	-	0
F2	S1	-	-	0
	S2	-	-	0
	S3	Q11	(1*30)*200	6000
	S4	Q11	(1*30)*200	6000
F3	S1	Q6	(2*25)*200	10000
	S2	-	-	0
	S3	-	-	0
	S4	Q6	(2*25)*200	10000
F4	S1	Q3	(2*35)*100	7000
	S2	Q3	(2*35)*100	7000
	S3	Q3	(2*35)*100	7000
	S4	-	-	0
F5	S1	Q6	(1*25)*300	7500
	S2	-	-	0
	S3	-	-	0
	S4	Q6	(1*25)*300	7500
F6	S1	Q3	(1*35)*100	3500
	S2	Q3	(1*35)*100	3500
	S3	Q3	(1*35)*100	3500
	S4	-	-	0

F7	S1	Q8, Q10	(3*30+3*40)*300	63000
	S2	Q1, Q8	(2*30+3*30)*300	45000
	S3	Q1, Q10	(2*30+3*40)*300	54000
	S4	Q8, Q10	(3*30+3*40)*300	63000
F8	S1	Q4	(2*20)*350	14000
	S2	Q4	(2*20)*350	14000
	S3	-	0	0
	S4	Q4	(2*20)*350	14000
F9	S1	Q7	(2*45)*150	13500
	S2	Q7, Q9	(2*45+3*10)*150	18000
	S3	Q7, Q9	(2*45+3*10)*150	18000
	S4	-	0	0
F10	S1	Q4	(2*20)*120	4800
	S2	Q4, Q5, Q9	(2*20+4*30+2*10)* 120	21600
	S3	Q5, Q9	(4*30+2*10)*120	16800
	S4	Q4, Q5	(2*20+4*30)*120	19200
F11	S1	-	0	0
	S2	-	0	0
	S3	-	0	0
	S4	-	0	0
F12	S1	Q10	(3*40)*100	12000
	S2	-	0	0
	S3	Q10	(3*40)*100	12000
	S4	Q10	(3*40)*100	12000

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

Site	Fragment
S1	F1,F3,F4,F5,F6,F7,F8F9,F10,F12,F11
S2	F4,F6,F7,F8,F9,F10
S3	F1,F2,F4,F6,F7,F9,F10,F12
S4	F2,F3,F5,F7,F8,F10,F12,F11

- For Fragment 11 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 11 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 block size = 512 Bytes
- Block pointer size = 4 Bytes
- Average Seek Time(S) = 4ms
- Average Disk Rotation(Latency) = 6 ms
- Block transfer rate(Tr) = 1 ms
- Fixed-length records are considered for all relations.
- Maximum records per relation = 2000
- Blocking Factor(bfr) = Floor(Block size / Record size)
- No of blocks required = Ceil(no of records / bfr)

Fragment	Relations	No. of records	Record size (Bytes)	Total size	Blocking factor	No. of blocks
Fragment 1	Carrier 1	100	30	3000	17	6
Fragment 2	Carrier 2	250	30	7500	17	15
Fragment 3	Genre 1	100	20	2000	25	4
Fragment 4	Genre 2	250	20	5000	25	10
Fragment 5	Media 1	400	32	12800	16	25
Fragment 6	Media 2	200	32	6400	16	13
Fragment 7	Product	1000	128	128000	4	250
Fragment 8	Subscription 1	500	20	10000	25	20

Fragment 9	Subscription 2	300	30	9000	17	18
Fragment 10	Customer	2000	64	128000	8	250
Fragment 11	Order details 1	500	64	32000	8	63
Fragment 12	Order details 2	500	32	16000	16	32

Storage Requirements:
Unspanned strategy is considered for storing records i.e. each record is stored in a single block. It helps in reducing block access at the expense of memory.

Indexing:
Primary indexing is used when the frequent queries are on the primary key of a relation.

Clustering indexing is used when the frequent queries are on non-key attribute of a relation.

Secondary indexing is used where primary or clustering indexing already exists but queries are accessing a non-key attribute very frequently. Then for that non-key attribute we used secondary indexing.

Fragment	Relation	Indexing type	Indexing attribute(Bytes)	Is it a key?
Fragment 1	Carrier 1	Clustering	rating(4)	No
Fragment 2	Carrier 2	Clustering	rating(4)	No
Fragment 3	Genre 1	Primary	genre_id(4)	Yes
Fragment 4	Genre 2	Primary	genre_id(4)	Yes
Fragment 5	Media 1	Primary	media_id(4)	Yes
Fragment 6	Media 2	Primary	media_id(4)	Yes

Fragment 7	Product	Clustering	name(6)	No
Fragment 8	Subscription 1	Primary	Subscription_id(4)	Yes
Fragment 9	Subscription 2	Primary	Subscription_id(4)	Yes
Fragment 10	Customer	Primary	email(6)	Yes
Fragment 11	Order details 1	Primary	order_id(4)	Yes
Fragment 12	Order details 2	Primary	order_id(4)	Yes

Comparing No. of disk access with and without indexing:

- No. of index records per block = ⌊Block size / Index file record size⌋
- No. of block access without indexing = No. of data blocks(Linear search)
- No. of block access with indexing:

  a. Primary Indexing = ⌈Log_2 (No. of index blocks) + 1⌉

  b. Cluster Indexing = ⌈Log_2 (No. of index blocks) + No. Of data blocks containing the required records⌉

Fragments	Relation	No of data records	No of data blocks	Index size per record	No. of index records per block	No. of index blocks	No of block access without indexing	No of block access with indexing
Fragment 1	Carrier 1	100	6	8	64	1	6	3
Fragment 2	Carrier 2	250	15	8	64	2	15	3
Fragment 3	Genre 1	100	4	8	64	2	4	2
Fragment 4	Genre 2	250	10	8	64	4	10	3

Fragment 5	Media 1	400	25	8	64	7	25	4
Fragment 6	Media 2	200	13	8	64	4	13	3
Fragment 7	Product	1000	250	10	51	10	250	2
Fragment 8	Subscription 1	500	20	8	64	8	20	4
Fragment 9	Subscription 2	300	18	8	64	5	18	4
Fragment 10	Customer	2000	250	12	42	48	250	7
Fragment 11	Order details 1	500	63	8	64	8	63	4
Fragment 12	Order details 2	500	32	8	64	8	32	4

Assume 50% are duplicates in case of clustering indexing.

Fragment 1 (Carrier 1):
No. of records = 100
No. of unique records = 50
No. of data blocks = 6
No. of index blocks = 1
Blocking factor = 17
Index blocking factor = 64

There will be only one index block consisting of 50 unique rating values. There are 3 records out of 50 which are spanning to 2 blocks.

Therefore, Average no. of block access = ceil(((50-3) * 2 + 3 * 3) / 50)
= 3

Fragment 2 (Carrier 2):
No. of records = 250
No. of unique records = 125
No. of data blocks = 15
No. of index blocks = 2
Blocking factor = 17
Index blocking factor = 64

There are 8 records out of 125 which are spanning to 2 blocks.
Therefore, average no. of block access = ceil(((125 - 8) * 2 + 8 * 3) / 125)

= 3

Fragment 7 (Product):
No. of records = 1000
No. of unique records = 500
No. of data blocks = 250
No. of index blocks = 10
Blocking factor = 4
Index blocking factor = 5
None of the records will span to more than one block.
Therefore, average no. of block access = 2

From the above table we can say that indexing reduces the number of block accesses needed to access a record in the data file.

Timings(hard disk) :

- Average Seek Time(S) is 4 ms irrespective of any site.
- Average Disk rotation time (Latency) Time (L) is 10 ms irrespective of any site.
- Suppose disk is rotating at 5000 rpm
- Time of one rotation = 60/5000 sec = 12ms
- Avg disk rotation time = (1/2)*one rotation time = 6ms
- Block transfer rate (Tr) is 1 ms irrespective of any site.
- Propagation Delay (Tp) = Distance between sites / Speed of transmission media
- Transmission Delay (Td) = Packet Size / Bandwidth
- Where, Speed of transmission media = 10^7 m/s
- Packet Size = 2000B
- Bandwidth = 10^6 B/s
- Td = 2ms

Local query access time = (seek time(4ms) + rotation time(6ms) + block transfer time(1ms)) * N

Local Update time = (seek time + rotation time + block transfer time) * N * 2
Where,

N is the number of disk block access(considering only using indexing).
N*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.

There are no update queries therefore local update time is zero.

Time required to access a record/table
= seek time + rotational time + block transfer time
= 4 + 6 + 1
= 11ms

Work Area Space and System Specification:

Here, we are measuring the maximum amount of buffer space required for computation.

According to our queries at a time maximum number of records that will be present in memory are of fragment customer and subscription consisting 2000 and 500 records respectively. Hence the total number of records are 2500.

Product record size is 128 Bytes and Subscription record size is 64B.
Total maximum buffer size required = 2000 * 128B + 500 * 64B
= 282 KB

All the relations have to be stored in disk. Total space required for storing all the relations is:
= 3000 + 7500 + 2000 + 5000 + 12800 + 6400 + 128000 + 10000 + 9000 + 128000 + 32000 + 16000
= 359700 Bytes

A disk containing around 352 KB free space would suffice for our database.