Problem Statement

A payroll management system is a system used by companies to help manage the computation, disbursement, and reporting of employees' salaries efficiently and accurately. It is the combination of software, processes, services, hardware, and other systems that help automate the payroll process from the gathering of timekeeping information, computation of wages, to disbursement of salaries and pay-slips.

Payroll Management System aids to manage employee information efficiently, define the deductions, leave etc., generate pay-slips at the convenience of a mouse click, generate and manage the payroll processes according to the salary structure assigned to the employee, generate all the reports related to the employee, attendance/leave, payroll etc.

The leave and attendance system maintains the employee working hour details of check in, check out, total working hours, effective working hours, paid and unpaid leaves taken by employee in the current month and extra days worked for which compensatory work off should be given. Depending on the organization policy, the system calculates the total number of hours in a month that the employee should be paid for. The leave and attendance system is a part of this organization itself.

The transport agency maintains information about the travel routes of employees, number of days in a month that the employee has availed the transport services and fare for the route in which the employee has traveled. The transport agency is an independent organization that maintains transport related information of not only the organization's employees being discussed, but also of other organizations' employees and individuals.

The food supplier maintains information about the bills for the food items procured by the employee during the month, percentage of some amount of which will be deducted from the employee's salary, if employee is working on that day, and complete deduction if employee was on leave. The food supplier is an independent module that provides food supplies to not only the organization's employees being discussed, but also to other organizations' employees and individuals.

The medical insurance provider maintains information about the hospital bills and medicine charges for the month, of the employee as well as his/her dependents. Depending on organization policies, a percentage of the total medical expense will be deducted from the salary. The medical insurance provider is an independent organization that provides medical insurance to not only the organization's employees being discussed, but also to other organizations' employees and individuals.

The payroll management system accumulates information from all the above mentioned systems and calculates the employee's gross pay, total deductions as per the organization policies and generates the pay-slip for the respective employee.

However, all these information are not maintained in a single data source because different sector maintains their respective information systems. Therefore, distributed scenario comes into picture that allows control and data flow between multiple data sources to the payroll management system in order to share the information. Based on the requirement, four data sources are identified.

The data sources are:

i. Leave and attendance system
ii. Transport agency
iii. Food Supplier
iv. Medical Insurance provider

ACTORS

The people who interact with the database:

i. Food supplier admin
ii. Transport agency admin
iii. Payroll admin
iv. Employees

SOME SAMPLE QUERIES

Complex Queries:

i. The payroll admin can get the food bill of the employee for the days he/she did not come to office (in order to completely deduct the amount from the salary) – We can get the working days of the employee from the leave and attendance system and check for the food bill from the food supplier for those days.
ii. An employee can get his/her monthly expenses – We can combine the expenses from transport agency, food supplier and medical insurance provider respectively.
iii. An employee can get his/her salary calculated – We can deduct the expenses from transport agency, food supplier and medical insurance provider respectively from the total pay based on working hours that we get from leave and attendance system.
iv. An employee can get his/her monthly travel and food expense for the days he/she worked - We can get the working days of the employee from the leave and attendance system and check for the travel fare and food bill from the transport agency and food supplier for those days respectively.

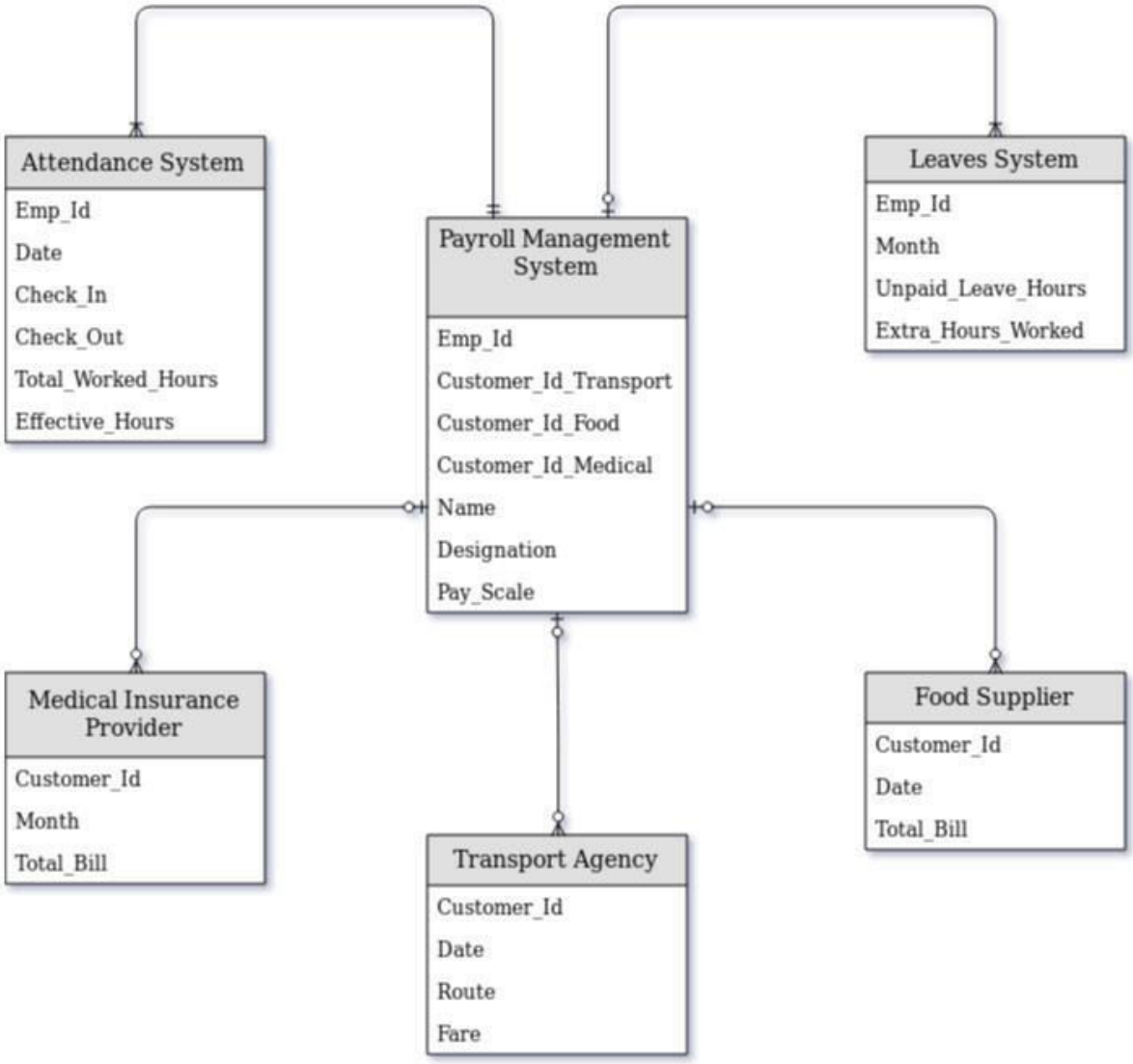
Simple Queries:

i. Food supplier admin can get for how many days of the month the employee is on planned leave so that food is not needed by him/her at the office premises – We can get the employee's planned leaves from leave and attendance system.
ii. Employee can get the medical expense for the current month – We can get the total bill from the medical insurance provider.
iii. Employee can get the total effective hours he/she has worked for in the current month to know if salary will be deducted for same – We can get the total effective hours from leave and attendance system.
iv. Employee can get the travel expense for the current month – We can get the total travel expense bill from the transport agency.

DATA SOURCES MAINTAINED AT DIFFERENT LOCATIONS

DATA SOURCE	LOCATION	INFORMATION
Leave and attendance system	Site A	Maintains employees' check in, check out time, total worked hours, effective hours on a regular basis. Also maintains record of unpaid leaves availed and extra hours worked by employee in the current month
Transport Agency	Site B	Maintains the travel routes of employees and the fare on a regular basis
Food Supplier	Site C	Maintains the food bill of the employee on a regular basis
Medical Insurance Provider	Site D	Maintains the medical bills of the employee for the current month

EER MODEL (CONCEPTUAL MODEL)



GLOBAL CONCEPTUAL SCHEMA

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

Attendance System:

Emp_Id	Date	Check_In	Check_Out	Total_ Worked_ Hours	Effective_Hours

Leaves System:

Emp_Id	Month	Unpaid_Leave_Hours	Extra_Hours_Worked

Transport Agency:

Customer_Id	Date	Route	Fare

Food Supplier:

Customer_Id	Date	Total_Bill

Medical Insurance Provider:

Customer_Id	Month	Total_Bill

Payroll Management System:

Emp _Id	Customer_Id_ Transport	Customer_ Id_ Food	Customer_ Id_Medical	Name	Designation	Pay_Scale

NORMALIZATION

Normalization is the process of organizing a database to reduce redundancy and improve data integrity. Normalization also simplifies the database design so that it achieves the optimal structure composed of atomic elements (i.e. elements that cannot be broken down into smaller parts).

Also referred to as database normalization or data normalization, normalization is an important part of relational database design, as it helps with the speed, accuracy, and efficiency of the database.

By normalizing a database, data can be arranged into tables and columns. It ensures that each table contains only related data. If data is not directly related, a new table can be created for that data.

For example, if we have a “Customers” table, we’d normally create a separate table for the products they can order (we could call this table “Products”). We’d create another table for customers’ orders (perhaps called “Orders”). And if each order could contain multiple items, we’d typically create yet another table to store each order item (perhaps called “OrderItems”). All these tables would be linked by their primary key, which allows us to find related data across all these tables (such as all orders by a given customer).

Normalization reduces redundancy and anomalies.

Redundancy : Redundancy means having repeated values in the same table, which leads to wastage of space, for example, for 100 employees working for the same department, we have to repeat the same department information for all these 100 employees in a table. We have huge storage capacity at lower cost nowadays. So, what is the issue here? Anomalies.

Anomalies : Normalization is the process of splitting relations into well structured relations that allow users to insert, delete, and update tuples without introducing database inconsistencies. Without normalization many problems can occur when trying to load an integrated conceptual model into the DBMS. These problems arise from relations that are generated directly from user views and are called anomalies.

Insertion Anomaly : An insertion anomaly is the inability to add data to the database due to absence of other data. For example, a Student_Group is defined so that null values are not allowed. If a new employee is hired but not immediately assigned to a Student_Group then this employee could not be entered into the database. This results in database inconsistencies due to omission.

Deletion Anomaly : A deletion anomaly is the unintended loss of data due to deletion of other data. For example, if the student group G1 was deleted from the table Student_Group, some departments would cease to exist. This results in database inconsistencies and is an

example of how combining information that does not really belong together into one table can cause problems.

Update Anomaly : An update anomaly is a data inconsistency that results from data redundancy and a partial update. For example, each employee in a company has a department associated with them as well as the student group they participate in.

There are many steps to achieve normalization :

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

Each normal form follows its own set of rules. Let's discuss them with the help of Payroll 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 or 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".

LHS is called key and RHS is called values.

For example, if I say 3 in the attribute A, then it must return BC values. This is called functional dependency. The term "functional" implies group of attributes that determine another group of attributes. We can also call this as 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 with attributes.

There are 3 different kinds of FD's :

- **Trivial Functional Dependency :** What is present in RHS is already a part of LHS.

Example : A determines itself.

A→ A
A→AB

AB→A
In general this form is represent ed as X ⊇ Y . However, there are no useful operations using this Y . However, there are no useful operations using this kind of dependency.

- **Non-trivial Functional Dependency :** Given a value of an attribute in LHS, we get a unique value in RHS.

Example :

A→BC
A → D
AB→ CD

In general, the rule is X ∩Y = φ

- **Semi Non-trivial Functional Dependency :** It is not deriving completely new information.

Example :

AB→BC
ACD → EFCD

B is common on both sides
CD is common both sides

It is generally represented as X ∩Y ≠ φ and X is not a super-set of Y.

FD's are quite useful in identifying keys, identifying equivalences of FD's and finding minimal FD sets. There are rules applied on FD such as inference rules, reflexivity, transitivity, augmentation, union, etc.

Closure : How many attributes we 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 : They are the 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 treated as the primary key of a table.

Super key : Sometimes, candidate key with non-key attribute uniquely determine a table. Such key is called super key.

Primary Key : Minimal super key or candidate key which has less number 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 preserving.

1st Normal Form : A relation is said to be in first normal form if it satisfies the following :

- No multi-valued attribute
- No composite attribute
- Primary key identified

Here, the relationship is converted to either relation or a foreign key is used or relations are merged.

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

Attendance System:

Emp_Id	Date	Check_In	Check_Out	Total_ Worked_ Hours	Effective_Hours

Leaves System:

Emp_Id	Month	Unpaid_Leave_Hours	Extra_Hours_Worked

Transport Agency:

Customer_Id	Date	Route	Fare

Food Supplier:

Customer_Id	Date	Total_Bill

Medical Insurance Provider:

Customer_Id	Month	Total_Bill

Payroll Management System:

Emp_Id	Customer_Id_Transport	Customer_Id_Food	Customer_Id_Medical	Name	Designation	Pay_Scale

Outcome of 1st normalization:

- Primary key has been identified in each table using closure property
- Composite attributes has been resolved
- Multi-valued attributes has been resolved

2nd Normal Form : 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.

Prime attribute (part of candidate key that determines anything else), also called partial dependency is identified, and eliminated. Because, 2nd NF is based on Full Functional dependency (key should determine all other attributes in a table).

Foreign Keys are identified.

Attendance System:

Emp_Id	Date	Check_In	Check_Out	Total_Worked_Hours	Effective_Hours

Leaves System:

Emp_Id	Month	Unpaid_Leave_Hours	Extra_Hours_Worked

Transport Agency:

Customer_Id	Date	Route

Fare:

Route [Foreign Key that references Route attribute of Transport Agency]	Fare
abc	100
xyz	200
mno	150

Food Supplier:

Customer_Id	Date	Total_Bill

Medical Insurance Provider:

Customer_Id	Month	Total_Bill

Payroll Management System:

Emp _Id	Customer_Id_ Transport	Customer_ Id_ Food	Customer_ Id_Medical	Name	Designation	Pay_Scale

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
- 3rd NF should hold the condition that : if X → Y then either X should be a super key or Y should be a prime attribute. Following this condition will never allow transitive dependency.
- In the above created relations, for every X → Y, X is always the super key. Hence the relations are already in 3rd Normal Form.

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 relations satisfy BCNF.

GLOBAL SCHEMA

Attendance System:

Attribute Name	Attribute Type (Size in bytes)
Emp_Id	Char (20)
Date	Date
Check_In	Timestamp
Check_Out	Timestamp
Total_Worked_Hours	Int (2)
Effective_Hours	Int (2)

Leaves System:

Attribute Name	Attribute Type (Size in bytes)
Emp_Id	Char (20)
Month	Char (9)
Unpaid_Leave_Hours	Int (3)
Extra_Hours_Worked	Int (3)

Transport Agency:

Attribute Name	Attribute Type (Size in bytes)
Customer_Id	Char (20)
Date	Date
Route	Char (50)

Fare:

Attribute Name	Attribute Type (Size in bytes)
Route	Char (50)
Fare	Int (3)

Food Supplier:

Attribute Name	Attribute Type (Size in bytes)
Customer_Id	Char (20)
Date	Date
Total_Bill	Int (7)

Medical Insurance Provider:

Attribute Name	Attribute Type (Size in bytes)
Customer_Id	Char (20)
Month	Char (9)
Total_Bill	Int (7)

Payroll Management System:

Attribute Name	Attribute Type (Size in bytes)
Emp_Id	Char (20)
Customer_Id_Transport	Char (20)
Customer_Id_Food	Char (20)
Customer_Id_Medical	Char (20)
Name	Char (50)
Designation	Char (20)
Pay_Scale	Int (3)

FRAGMENTATION

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

- When storage exhausted out
- For parallel processing
- For load balancing
- To improve query response time
- For better local processing
- 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**
- **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
- **Disjointness** : Row or column must be present in at most one site, else it will lead to inconsistent data Fragmentation takes place in a relation based on the query and its frequency.

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 Payroll Management System :

1. **Get the food bill of the employee for the days he/she did not come to office:**

```
SELECT sum(Total_Bill) FROM Food_Supplier WHERE (Customer_Id, Date) IN
(SELECT Customer_Id_Food, Date FROM Payroll_Management_System p,
Attendance_System a WHERE a.Total_Worked_Hours = 0 AND p.Emp_Id =
'12345' AND a.Emp_Id = '12345' AND a.Date BETWEEN '2019-10-01' AND '2019-
10-31');
```

2. **Get monthly expense of the employee:**

```
SELECT
(SELECT sum(Fare) FROM Fare f, Transport_Agency t WHERE t.Customer_Id IN
(SELECT  Customer_Id_Transport FROM Payroll_Management_System WHERE
Emp_Id = '12345') AND t.Route = f.Route AND t.Date BETWEEN '2019-10-01'
AND   '2019-10-31')
+
(SELECT sum(Total_Bill) FROM Medical_Insurance_Provider WHERE Month =
'Oct2019' AND Customer_Id IN (SELECT Customer_Id_Medical FROM
Payroll_Management_System WHERE Emp_Id = '12345'))
+
(SELECT sum(Total_Bill) FROM Food_Supplier WHERE Customer_Id IN
(SELECT Customer_Id_Food FROM Payroll_Management_System WHERE
Emp_Id = '12345') AND Date BETWEEN '2019-10-01' AND '2019-10-31')
AS total_expense;
```

3. **Get salary of the employee:**

```
SELECT
```

```
(SELECT Pay_Scale FROM Payroll_Management_System WHERE Emp_Id =
'12345') / 12
-
(
((SELECT Unpaid_Leave_Hours FROM Leaves_System WHERE Emp_Id =
'12345' AND Month = 'Oct2019') * 100)
+
(SELECT sum(Fare) FROM Fare f, Transport_Agency t WHERE t.Customer_Id IN
(SELECT Customer_Id_Transport FROM Payroll_Management_System WHERE
Emp_Id = '12345') AND t.Route = f.Route AND t.Date BETWEEN '2019-10-01'
AND '2019-10-31')
+
(SELECT sum(Total_Bill) FROM Medical_Insurance_Provider WHERE Month =
'Oct2019' AND Customer_Id IN (SELECT Customer_Id_Medical FROM
Payroll_Management_System WHERE Emp_Id = '12345'))
+
(SELECT sum(Total_Bill) FROM Food_Supplier WHERE Customer_Id IN
(SELECT Customer_Id_Food FROM Payroll_Management_System WHERE
Emp_Id = '12345') AND Date BETWEEN '2019-10-01' AND '2019-10-31')
)
+
((SELECT Extra_Hours_Worked FROM Leaves_System WHERE Emp_Id =
'12345' AND Month = 'Oct2019') * 100)
AS salary;
```

4. **Get monthly food and travel expense of the employee for the days worked:**

```
SELECT
(SELECT sum(Total_Bill) FROM Food_Supplier WHERE (Customer_Id, Date) IN
(SELECT Customer_Id_Food, Date FROM Payroll_Management_System p,
Attendance_System a WHERE a.Total_Worked_Hours >= 9 AND p.Emp_Id =
'12345' AND a.Emp_Id = '12345' AND a.Date BETWEEN '2019-10-01' AND '2019-
10-31'))
+
(SELECT sum(Fare) FROM Fare f, Transport_Agency t WHERE (t.Customer_Id,
t.Date) IN (SELECT Customer_Id_Transport, Date FROM
Payroll_Management_System p, Attendance_System a WHERE
a.Total_Worked_Hours >= 9 AND p.Emp_Id = '12345' AND a.Emp_Id = '12345'
AND a.Date BETWEEN '2019-10-01' AND '2019- 10-31') AND t.Route = f.Route)
AS food_and_travel_expense;
```

5. **Get days of month when the employee will be on planned leave:**

```
SELECT
(SELECT Extra_Hours_Worked FROM Leaves_System WHERE Emp_Id = '12345'
AND Month = 'Nov2019') / 9
AS planned_leave_days;
```

6. **Get the medical expense of the employee for the current month:**

SELECT sum(Total_Bill) FROM Medical_Insurance_Provider WHERE Month = 'Oct2019' AND Customer_Id IN (SELECT Customer_Id_Medical FROM Payroll_Management_System WHERE Emp_Id = '12345');

7. **Get the total effective hours of the employee for the current month:**

   SELECT Effective_Hours FROM Attendance_System WHERE Emp_Id = '12345' AND Date BETWEEN '2019-10-01' AND '2019-10-31';

8. **Get the travel expense of the employee for the current month:**

   SELECT sum(Fare) FROM Fare f, Transport_Agency t WHERE t.Date BETWEEN '2019-10-01' AND '2019-10-31' AND t.Customer_Id IN (SELECT Customer_Id_Transport FROM Payroll_Management_System WHERE Emp_Id = '12345') AND t.Route = f.Route ;

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 queries are 2nd and 3rd ones as mostly employees will be interested in their monthly expenses and calculated salaries respectively. Therefore, based on the frequency of the queries, the predicates are:

1. **Customer_Id = 'value' and Route = 'value' and Date = 'range' from Transport_Agency relation**

Example : Customer_Id = 'CT12345' AND Route = 'xyz' AND Date BETWEEN '2019-10-01' AND '2019-10-31'

2. **Route = 'value' from Fare relation**
   Example : Route = 'xyz'

3. **Customer_Id = 'value' and Date = 'range' from Food_Supplier relation**
   Example : Customer_Id = 'CF12345' AND Date BETWEEN '2019-10-01' AND '2019-10-31'

4. **Customer_Id = 'value' and Month = 'value' from Medical_Insurance_Provider relation**
   Example : Customer_Id = 'CM12345' AND Month = 'Oct2019'

5. **Emp_Id = 'value' and Month = 'value' from Leaves_System**
   Example : Emp_Id = '12345' AND Month = 'Oct2019'

6. **Emp_Id = 'value' and Customer_Id_Transport = 'value' and Customer_Id_Food = 'value' and Customer_Id_Medical = 'value' and Pay_Scale = 'value' from Payroll_Management_System relation**

Example : Emp_Id = '12345' AND Customer_Id_Transport = 'CT12345' AND Customer_Id_Food = 'CF12345' and Customer_Id_Medical = 'CM12345' and Pay_Scale = '240000'

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:

- 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.
- 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.

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: Attendance_System, Leaves_System, Transport_Agency, Fare, Food_Supplier, Medical_Insurance_Provider, Payroll_Management_System

Relation 1 : Attendance_System

Attribute Usage Matrix:

Query No.	Emp_Id	Date	Check_ In	Check_ Out	Total_ Worked_ Hours	Effective_Hours
Q1	1	1	0	0	1	0
Q2	0	0	0	0	0	0
Q3	0	0	0	0	0	0
Q4	1	1	0	0	1	0
Q5	0	0	0	0	0	0
Q6	0	0	0	0	0	0
Q7	1	1	0	0	0	1
Q8	0	0	0	0	0	0

Query Access frequency matrix :-

Sites	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8
Leave and Attendance System	0	0	0	0	23	0	18	0
Transport Agency	0	0	0	0	32	0	0	29
Food Supplier	0	0	0	0	35	0	0	0
Medical Insurance Provider	0	0	0	0	0	44	0	0
Payroll Management System	20	45	68	10	0	36	25	37

Attribute affinity matrix :-

	Emp_Id	Date	Check_In	Check_ Out	Total_ Worked_ Hours	Effective_ Hours
Emp_Id	73	73	0	0	30	43
Date	73	73	0	0	30	30
Check_In	0	0	0	0	0	0
Check_Out	0	0	0	0	0	0
Total_ Worked_ Hours	30	30	0	0	30	0
Effective_ Hours	43	43	0	0	0	43

For Check_In :

CONT(A0, Check_In, Emp_Id) = 0 + 0 – 0 = 0
CONT(Emp_Id, Check_In, Date) = 0 + 0 – 16448 = -16448
CONT(Date, Check_In, Check_Out) = 0 + 0 – 0 = 0

Hence, CONT(Date, Check_In, Check_Out) is more.

Attributes : Emp_Id, Date, Check_In

For Check_Out :

CONT(A0, Check_Out, Emp_Id) = 0 + 0 – 0 = 0
CONT(Emp_Id, Check_Out, Date) = 0 + 0 – 16448 = -16448
CONT(Date, Check_Out, Check_In) = 0 + 0 – 0 = 0
CONT(Check_In, Check_Out, Total_Worked_Hours) = 0 + 0 – 0 = 0

Hence, CONT(Check_In, Check_Out, Total_Worked_Hours) is more.

Attributes : Emp_Id, Date, Check_In, Check_Out

For Total_Worked_Hours :

CONT(A0, Total_Worked_Hours, Emp_Id) = 0 + 10560 – 0 = 10560
CONT(Emp_Id, Total_Worked_Hours, Date) = 10560 + 10560 – 16448 = 4672
CONT(Date, Total_Worked_Hours, Check_In) = 10560 + 0 – 0 = 10560
CONT(Check_In, Total_Worked_Hours, Check_Out) = 0 + 0 – 0 =0
CONT(Check_Out, Total_Worked_Hours, Effective_Hours) = 0 + 4380 – 0 = 4380

Hence, CONT(Date, Total_Worked_Hours, Check_In) is more.

Attributes : Emp_Id, Date, Total_Worked_Hours, Check_In, Check_Out

For Effective_Hours :

CONT(A0, Effective_Hours, Emp_Id) = 0 + 14356 – 0 = 14356
CONT(Emp_Id, Effective_Hours, Date) = 14356 + 14356 – 16448 = 12264
CONT(Date, Effective_Hours, Total_Worked_Hours) = 14356 + 4380 – 10560 = 8176
CONT(Total_Worked_Hours, Effective_Hours, Check_In) = 10560 +0 – 0 = 10560
CONT(Check_In, Effective_Hours, Check_Out) = 0 + 0 – 0 =0
CONT(Check_Out, Effective_Hours, A7) = 0 + 0 – 0 = 0

Hence, CONT(A0, Effective_Hours, Emp_Id) is more.

Attributes : Effective_Hours , Emp_Id, Date, Total_Worked_Hours, Check_In, Check_Out

Partitions : {Effective_Hours} , {Emp_Id, Date, Total_Worked_Hours, Check_In, Check_Out}

z = 0 * 30 – 1849 = -1849

Partitions : {Effective_Hours , Emp_Id} , {Date, Total_Worked_Hours, Check_In, Check_Out}

z = 0 * 0 – 5329 = -5329

Partitions : {Effective_Hours , Emp_Id, Date} , {Total_Worked_Hours, Check_In, Check_Out}

z = 43 * 0 – 900 = -900

Partitions : {Effective_Hours , Emp_Id, Date, Total_Worked_Hours} , {Check_In, Check_Out}

z = 5329 * 0 – 0 = 0

Partitions : {Effective_Hours , Emp_Id, Date, Total_Worked_Hours, Check_In} , {Check_Out}

z = 5329 * 0 – 0 = 0

Hence we get non-negative value for :

Partitions : {Effective_Hours , Emp_Id, Date, Total_Worked_Hours} and {Check_In, Check_Out}

Relation 2 : Leaves_System

Attribute Usage Matrix:

Query No.	Emp_Id	Month	Unpaid_Leave_Hours	Extra_Hours_Worked

Q1	0	0	0	0
Q2	0	0	0	0
Q3	1	1	1	1
Q4	0	0	0	0
Q5	1	1	0	1
Q6	0	0	0	0
Q7	0	0	0	0
Q8	0	0	0	0

Query Access frequency matrix :-

Sites	**Q1**	**Q2**	**Q3**	**Q4**	**Q5**	**Q6**	**Q7**	**Q8**
Leave and Attendance System	0	0	0	0	23	0	18	0
Transport Agency	0	0	0	0	32	0	0	29
Food Supplier	0	0	0	0	35	0	0	0
Medical Insurance Provider	0	0	0	0	0	44	0	0
Payroll Management System	20	45	68	10	0	36	25	37

Attribute affinity matrix :-

	Emp_Id	**Month**	**Unpaid_Leave_Hours**	**Extra_Hours_Worked**
Emp_Id	158	158	68	158
Month	158	158	68	158
Unpaid_ Leave_ Hours	68	68	68	68
Extra_ Hours_ Worked	158	158	68	158

For Unpaid_Leave_Hours :

CONT(A0, Unpaid_Leave_Hours, Emp_Id) = 0 + 102152 – 0 = 102152
CONT(Emp_Id, Unpaid_Leave_Hours, Month) = 102152 + 102152 – 159032 = 45272
CONT(Month, Unpaid_Leave_Hours, Extra_Hours_Worked) = 102152 + 102152 - 159032 = 45272

Hence, CONT(Month, Unpaid_Leave_Hours, Extra_Hours_Worked) is more.

Attributes : Emp_Id, Month, Unpaid_Leave_Hours

For Extra_Hours_Worked :

CONT(A0, Extra_Hours_Worked, Emp_Id) = 0 + 159032 – 0 = 159032
CONT(Emp_Id , Extra_Hours_Worked, Month) = 159032 + 159032 - 159032 = 159032
CONT(Month, Extra_Hours_Worked, Unpaid_Leave_Hours) = 159032 + 102152 - 102152 = 159032
CONT(Unpaid_Leave_Hours, Extra_Hours_Worked, A5) = 102152 + 0 – 0 = 102152

Hence, CONT(Month, Extra_Hours_Worked, Unpaid_Leave_Hours) is more.

Attributes : Emp_Id, Month, Extra_Hours_Worked, Unpaid_Leave_Hours

Partitions : {Emp_Id} , {Month, Extra_Hours_Worked, Unpaid_Leave_Hours}

z = 0 * 0 – 24964 = -24964

Partitions : {Emp_Id, Month} , {Extra_Hours_Worked, Unpaid_Leave_Hours}

z = 0 * 0 – 24964 = -24964

Partitions : {Emp_Id, Month, Extra_Hours_Worked} , {Unpaid_Leave_Hours}

z = 8100 * 0 – 4624 = -4624

As all are negative values, so partitioning is not needed.

{Emp_Id, Month, Extra_Hours_Worked, Unpaid_Leave_Hours}

Relation 3 : Transport_Agency

Attribute Usage Matrix:

Query No.	Customer_Id	Date	Route
Q1	0	0	0
Q2	1	1	1

Q3	1	1	1
Q4	1	1	1
Q5	0	0	0
Q6	0	0	0
Q7	0	0	0
Q8	1	1	1

Query Access frequency matrix :-

Sites	**Q1**	**Q2**	**Q3**	**Q4**	**Q5**	**Q6**	**Q7**	**Q8**
Leave and Attendance System	0	0	0	0	23	0	18	0
Transport Agency	0	0	0	0	32	0	0	29
Food Supplier	0	0	0	0	35	0	0	0
Medical Insurance Provider	0	0	0	0	0	44	0	0
Payroll Management System	20	45	68	10	0	36	25	37

Attribute affinity matrix :-

	Customer_Id	**Date**	**Route**
Customer_ Id	189	189	189
Date	189	189	189
Route	189	189	189

For Route :

CONT(A0, Route, Customer_Id) = 0 + 214326 – 0 = 214326

CONT(Customer_Id, Route, Date) = 214326 + 214326 – 214326 = 214326

CONT(Date, Route, A4) = 214326 + 0 – 0 = 214326

As we get same positive value for all CONT, we can keep Route as it is.

Attributes : Customer_Id, Date , Route

Partitions : {Customer_Id} , {Date , Route}

z = 0 * 0 – 35721 = -35721

Partitions : {Customer_Id, Date} , {Route}

z = 0 * 0 – 35721 = -35721

As both z values are negative, so no need to partition here.

{Customer_Id, Date , Route}

Relation 4 : Fare

Attribute Usage Matrix:

Query No.	Route	Fare
Q1	0	0
Q2	1	1
Q3	1	1
Q4	1	1
Q5	0	0
Q6	0	0
Q7	0	0
Q8	1	1

Query Access frequency matrix :-

Sites	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8
Leave and Attendance System	0	0	0	0	23	0	18	0
Transport Agency	0	0	0	0	32	0	0	29
Food Supplier	0	0	0	0	35	0	0	0
Medical Insurance Provider	0	0	0	0	0	44	0	0
Payroll Management System	20	45	68	10	0	36	25	37

Attribute affinity matrix :-

	Route	Fare
Route	189	189
Fare	189	189

As only two attributes are there, so there is no need of partitioning.

{Route, Fare}

PHYSICAL DESIGN

This physical design talks about how these fragments are stored in secondary memory. Based on global schema defined in a section above, 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	No. of Records	Record Size in bytes	Blocking Factor	No. of blocks
1	Attendance System	50	14	73	1
2	Leaves System	50	22	46	2
3	Transport Agency	200	48	21	10
4	Payroll Management System	100	48	21	5
5	Payroll Management System	500	86	11	46
6	Payroll Management System	300	48	21	15
7	Food Supplier	200	48	21	10
8	Attendance System	150	48	21	8
9	Payroll Management System	50	22	46	2
10	Transport Agency	200	4	256	1
11	Payroll Management System	5000	65	15	334
12	Leaves System	7000	10	102	69
13	Food Supplier	200	86	11	19
14	Transport Agency	200	36	28	8

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.

Fragments 1 and 10 are fit into one block. Therefore, using indexing is not really needed.

Fragment	Relation	Indexing Type	Indexing Attribute(s)	Is a key?
1	Attendance System	NA	NA	No
2	Leaves System	Cluster	Emp_Id	No
3	Transport Agency	Cluster	Customer_Id	No
4	Payroll Management System	Cluster	Emp_Id	Yes
5	Payroll Management System	Cluster	Emp_Id	Yes
6	Payroll Management System	Cluster	Emp_Id	Yes
7	Food Supplier	Cluster	Customer_Id	No
8	Attendance System	Cluster	Emp_Id	No
9	Payroll Management System	Cluster	Emp_Id	Yes
10	Transport Agency	NA	NA	NA
11	Payroll Management System	Primary	Emp_Id	Yes
12	Leaves System	Primary	Emp_Id, Month	Yes
13	Food Supplier	Cluster	Customer_Id	No
14	Transport Agency	Primary	Customer_Id, Date	Yes

Fragment	Relation	No. of 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
1	Atten-dance System	50	1	NA	NA	NA	1	1
2	Leaves System	50	2	4+4	124	1	2	1
3	Transport Agency	200	10	4+10	73	3	10	3
4	Payroll	100	5	4+10	73	2	5	2

	Manage-ment System							
5	Payroll Manage-ment System	500	46	4+20	42	12	46	5
6	Payroll Manage-ment System	300	15	4+10	73	5	15	3
7	Food Supplier	200	10	4+10	73	3	10	3
8	Atten-dance System	150	8	4+10	73	3	8	3
9	Payroll Manage-ment System	50	2	4+4	124	1	2	1
10	Transport Agency	200	1	NA	NA	NA	1	1
11	Payroll Manage-ment System	5000	334	4+4	124	41	334	6
12	Leaves System	7000	69	4+4	124	55	69	7
13	Food Supplier	200	19	4+20	42	5	19	4
14	Transport Agency	200	8	4+4	124	2	8	2

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 shown 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 :

i. Local Query Time = (Seek Time + Latency + Block Transfer Time) * N
ii. 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 no. 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	No. of Records	No. of data blocks	No. of block access with indexing	Local Query Time (ms) = (S + L + Tr) * N	Remote Query Time (ms)
1	Attendance System	50	1	1	30.5	132.5
2	Leaves System	50	2	1	30.5	132.5
3	Transport Agency	200	10	3	91.5	193.5
4	Payroll Management System	100	5	2	61	163
5	Payroll Management System	500	46	5	152.5	254
6	Payroll Management System	300	15	3	91.5	193.5
7	Food Supplier	200	10	3	91.5	193.5
8	Attendance System	150	8	3	91.5	193.5
9	Payroll Management System	50	2	1	30.5	132.5
10	Transport Agency	200	1	1	30.5	132.5
11	Payroll Management System	5000	334	6	183	285

12	Leaves System	7000	69	7	213.5	315.5
13	Food Supplier	200	19	4	122	224
14	Transport Agency	200	8	2	61	163

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,

Packet Transmission Time = (Size of packet) / Bandwidth = (1500 B) / (10^6 B/s) = 0.0015s , approximately equal to 2ms

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

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.

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 Site	Frequency	Fragment Access
Q1	S1	200	F1 - 4R
Q1	S2	300	F2 - 7R
Q1	S3	400	F3 – 3R
Q1	S4	500	F4 - 2R
Q2	S1	250	F5 - 8R
Q2	S2	150	F6 - 4R
Q2	S3	50	F7 - 6R
Q2	S4	100	F8 - 2R

Q3	S1	200	F9 - 3R
Q3	S2	400	F10 - 4R
Q3	S3	600	F11 - 6R
Q3	S4	800	F12 - 3R
Q4	S1	900	F1 - 4R
Q4	S2	300	F2 - 7R
Q4	S3	200	F3 – 3R
Q4	S4	100	F4 - 2R
Q5	S1	150	F5 - 8R
Q5	S2	200	F6 - 4R
Q5	S3	250	F7 - 6R
Q5	S4	100	F8 - 2R
Q6	S1	250	F9 - 3R
Q6	S2	350	F10 - 4R
Q6	S3	600	F11 - 6R
Q6	S4	300	F12 - 3R
Q7	S1	400	F1 - 4R
Q7	S2	100	F2 - 7R
Q7	S3	150	F3 – 3R
Q7	S4	350	F4 - 2R
Q8	S1	300	F5 - 8R
Q8	S2	200	F6 - 4R
Q8	S3	500	F7 - 6R
Q8	S4	300	F8 - 2R

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**	**Query**	**No. of Reads * Frequency * (Remote Time – Local Time)**	**Benefit (ms)**
F1	S1	Q1, Q4, Q7	200 * 4 * 102 + 300 * 4 * 102+	367200

			400 * 4 * 102	
F1	S2	none	none	0
F1	S3	none	none	0
F1	S4	none	none	0
F2	S1	none	none	0
F2	S2	Q1, Q4, Q7	900 * 7 * 102 + 300 * 7 * 102 + 200 * 7 * 102	1071000
F2	S3	none	none	0
F2	S4	none	none	0
F3	S1	none	none	0
F3	S2	none	none	0
F3	S3	Q1, Q4, Q7	400 * 3 * 102+ 100 * 3 * 102+ 150 * 3 * 102	198900
F3	S4	none	none	0
F4	S1	none	none	0
F4	S2	none	none	0
F4	S3	none	none	0
F4	S4	Q1, Q4, Q7	500 * 2 * 102+ 100 * 2 * 102+ 350 * 2 * 102	193800
F5	S1	Q2, Q5, Q8	250 * 8 * 101.5 + 150 * 8 * 101.5+ 300 * 8 * 101.5	568400
F5	S2	none	none	0
F5	S3	none	none	0
F5	S4	none	none	0
F6	S1	none	none	0
F6	S2	Q2, Q5, Q8	150 * 4 * 102+ 200 * 4 * 102+ 200 * 4 * 102	224400
F6	S3	none	none	0
F6	S4	none	none	0
F7	S1	none	none	0
F7	S2	none	none	0

F7	S3	Q2, Q5, Q8	50 * 6 * 102+ 250 * 6 * 102+ 500 * 6 * 102	489600
F7	S4	none	none	0
F8	S1	none	none	0
F8	S2	none	none	0
F8	S3	none	none	0
F8	S4	Q2, Q5, Q8	100 * 2 * 102+ 100 * 2 * 102+ 300 * 2 * 102	102000
F9	S1	Q3, Q6	200 * 3 * 101.7+ 250 * 3 * 101.7	137295
F9	S2	none	none	0
F9	S3	none	none	0
F9	S4	none	none	0
F10	S1	none	none	0
F10	S2	Q3, Q6	400 * 4 * 101.7+ 350 * 4 * 101.7	305100
F10	S3	none	none	0
F10	S4	none	none	0
F11	S1	none	none	0
F11	S2	none	none	0
F11	S3	Q3, Q6	600 * 6 * 102 + 600 * 6 * 102	734400
F11	S4	none	none	0
F12	S1	none	none	0
F12	S2	none	none	0
F12	S3	none	none	0
F12	S4	Q3, Q6	800 * 3 * 102+ 300 * 3 * 102	336600
F13	S1	none	none	0
F13	S2	none	none	0
F13	S3	none	none	0
F13	S4	none	none	0

So based on the Benefit table, we can infer the following:

1. F1 is put at S1
2. F2 is put at S2
3. F3 is put at S3
4. F4 is put at S4
5. F5 is put at S1
6. F6 is put at S2
7. F7 is put at S3
8. F8 is put at S4
9. F9 is put at S1
10. F10 is put at S2
11. F11 is put at S3
12. F12 is put at S4

INDEXING

A database index is a data structure that improves the speed of data retrieval operations on a database table at the cost of additional writes and storage space to maintain the index data structure. Indexes are used to quickly locate data without having to search every row in a database table every time a database table is accessed.

Indexing is defined based on its indexing attributes. Indexing can be of the following types –

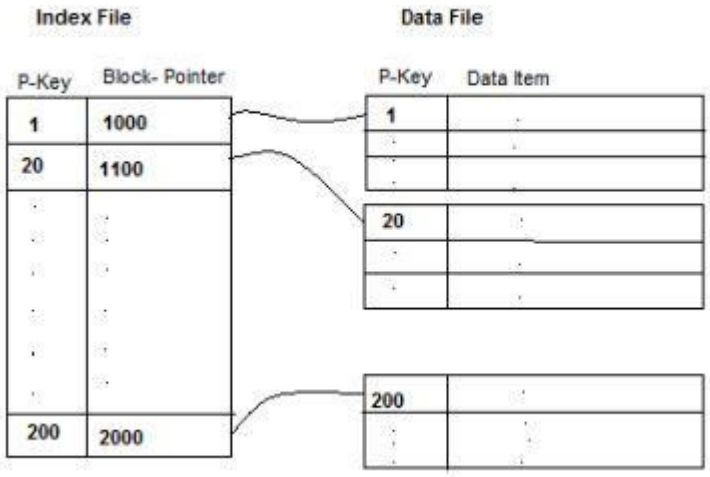
- Primary Index − Primary index is defined on an ordered data file. The data file is ordered on a key field. The key field is generally the primary key of the relation.
- Secondary Index − Secondary index may be generated from a field which is a candidate key and has a unique value in every record, or a non-key with duplicate values.
- Clustering Index − Clustering index is defined on an ordered data file. The data file is ordered on a non-key field.

In our assumption, we have not considered indexing. Therefore, to access a fragment, all the blocks have to be accessed.

For ex, to do a search in fragment F1, in the worst case, all of its blocks has to be accessed. That is a major flaw of not using indexing. So, we will go for indexing now. In the proposed system,

most of the queries are accessing the tables using the primary key attribute. So, it is profitable to index the tables on the primary key.

Therefore, we are going for Primary Indexing.



Assuming,
Block Pointer Size = 6B
Block Size = 1024B

Fragments	No. of blocks	Size of index table entry	Total size of index table	No. of blocks
F1	74	10+6	1184	2
F2	206	10+6	3296	4
F3	4	10+6	64	1
F4	352	10+6	5632	6
F5	59	10+6	944	1
F6	118	10+6	1888	2
F7	20	10+6	320	1

Hence from here we can see that we have reduced the no of block access required for fragment 4 from 352 to 7 using primary indexing. In the worst case we will have to access 7 blocks. 6 for the index blocks + 1 for the data block. Therefore it is an improvement in number of accessing blocks.

Same is the case with other Fragments.