|  |
| --- |
| FARMER INFORMATION SYSTEM |
| DBMS Mini Project |
| kvb |

Praveen Jain 07IT36

Veeresh B 07IT64

Karthik V Bellur 07IT20

Contents

1. Problem Description 3
2. EER Diagram 7
3. Relations 8
4. Advanced logical design 9
5. Physical design 11

PROBLEM DESCRIPTION

# Introduction:

India is a developing country with large agriculture based economy. Agriculture being a primary occupation employs majority number of people. The core aspect of whole agricultural system is the farmer. Majority of farmer centric activities take place in rural areas. Also majority of the contributions in this sector come from small farmers. The new technological boom in Indian urban scenario has not touched the rural lives in effective ways. This project aims at bringing technology closer to these small and medium scale farmers in order to help them make timely and informed decisions.

The success of a crop for a farmer depends completely on the choice of crop, irrigation, seeds, fertilizers, pesticides, getting proper subsidies and market conditions. The current scheme of gaining this information is left to mercy of the local government agency (panchayat board) passing on this information. Also many farmers especially small farmers miss out on subsidy opportunities, loan schemes, market booms etc mainly due to lack of information. This project aims to bridge the information gap.

A central government body (such as central government) updates all the necessary and critical information such as crop details, subsidies on equipment and chemicals, financial schemes, soil and fertilizer analysis reports, market conditions. The farmer is able to view this information gathered from various sources and research institutes and make an informed choice. Also the information given to farmer is made specific and applicable to the region in which he has the land.

This kind of information surety helps farmers avoid losses and failed yield due to sowing of wrong crops, spraying wrong pesticides or lack of it, failing to capitalize on monetary / market conditions, etc. This system to large extent aims to help farmers get right price for the yield.

# Features:

* Provides a unique and hazel-free information flow from decision making government bodies to benefitting farmers.
* Helps flow of data collected from advanced technologies (such as weather forecasting) to reach farmers in time.
* A registration for every farmer based on existing ID scheme and land registration scheme. No additional ID / registration schemes need to be added. The farmer selects his land location form a preset locations covering the whole area. All further details are published to him based on this choice of location.
* Allows profile management for farmers with ability to upload previous crops and present sowed crops. The government can derive current crop statistics of a farmer from the database.
* Central government assured up-to-date information for farmers across the whole nation.
* Filtered and tailored information for each location and land type the farmer owns.
* Ability to manage multiple farms (can be of different type) by a farmer.
* Basic soil analysis on location basis and fertilizer recommendations. Also crop recommendations based on location and region from a system maintained Knowledge Bank.

# Activities:

The system activity based on the views / end users of the system:

**Farmer’s view**: Typically the farmer has to register with the system. The farmer is expected to provide personal details. Apart from this he has to select the location of his land. After successful creation of account the farmer can then login and upload rest of information. The farmers can typically browse through the local and national schemes and where to avail them. They are also presented with local and higher office information. They are also supposed to enter information relevant to their crops and from a Knowledge Bank maintained by the system they will be given knowledge on the scientific means of cultivation based on the details specified.

**Up loader / Data entry view**: The data entry personal are typically employed by the government. They can enter new crops, new locations, new fertilizers, and new financial schemes and update current schemes and values. They are also supposed to update the Knowledge Bank which is maintained by the system which is handy while giving farmers advices on the scientific methods of cultivation. They are typically presented with range of choices to enter / update any of above data. After choosing a data to enter / update a specialized window/form for that data is presented. It helps them enter all the required information. All associated information is also available in that window/form. Eg: while creating new crop all fertilizers are given there so that a crop can be associated with a fertilizer.

**Project Manager view**: The project manager is basically the head of the operation appointed by the government. He can add new (or delete existing) data entry users, create summarized reports.

## System Details:

The **farmer** plays the central role in the whole system. He has a unique (system specific)ID. He also has a name, address, a predefined location, a income, land details such as: area, fertility, type of soil. He can upload a set of previous crops sowed by him and current crop under irrigation.

Another central concept is the **crop** sowed (or to be sown) by the farmer. The crop again has a system specific ID, a name, a type (vegetable, herb, staple food, etc) ,its current market price, the price of seeds for this crop, suited soil and fertilizers details ,water requirement.

Chemicals necessary for modern agriculture include the **fertilizers and pesticides**. They have a name, manufacturer, system specific ID, a type attribute (to internally recognize as a pesticide / fertilizer. In user view these will be separated and displayed), the subsidized price details, approved distributor list

**Location** determines the possible weather conditions and availability of various schemes. A location has state, district, weather scenario, regional office address

**Farm equipment** plays crucial role in farmers output and efficiency. It has a name, manufacturer, subsidized price, approved distributors, a type (automated farm machinery, irrigation equipment, hand equipment, etc)

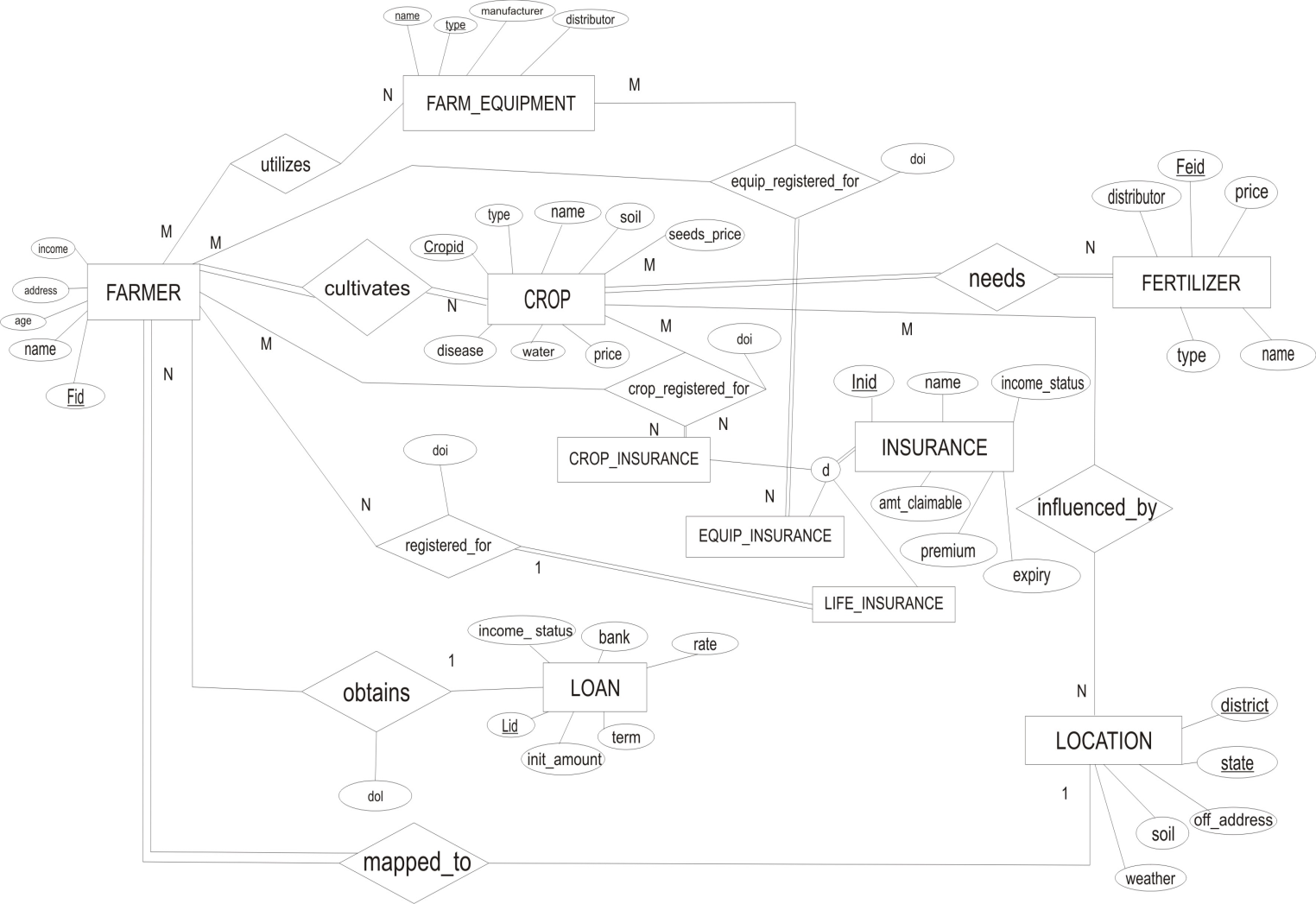
**Loan** schemes provide the required capital for buying seeds and equipment. It has a loan ID, bank giving loan; rate of interest, initial amount, term of the loan, income group which benefits from this.

**Crop Insurance** is critical in ensuring the farmers can recover from major crop failures this year. It has a unique ID, income group benefitting from this, name, annual premium, expiry (or renewal date), amount claimable.

# Query Details:

1. Farmer may want to view his information and update it (simple query).
2. Search a financial scheme or loan available to farmers by name / income range, etc (simple query).
3. Get information of a particular crop with suitable fertilizer and soil type(complex query).
4. Information on which crops are being cultivated based on income range of the farmer (simple query).
5. The data entry user can search for crops / fertilizers/ loan schemes to update / delete the data (simple query).
6. Information about which crop is suitable to be grown based on farmer’s location, soil type, weather condition (complex query).
7. Information on various insurance schemes which the farmers of a certain income range have availed for their farm equipment and crops (complex query).

EER DIAGRAM



RELATIONS

* Obtained from the EER Model

FARMER

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Fid | Name | Age | Address | Income | Doi | Dol | Inid | Lid | District | State |

CROP

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Cropid | Type | Name | Soil | Disease | Water | Price | Seeds\_price |

FERTILIZER

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Feid | Name | Type | Distributor | Price |

LOCATION

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| District | State | Weather | Soil | Off\_address |

LOAN

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Lid | Init\_amount | Term | Bank | Income\_status | Rate |

INSURANCE

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Inid | Name | Income\_status | Type | Premium | Expiry | Amt\_claimable |

FARM\_EQUIPMENT

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Manufacturer | Distributor |

UTILIZES

|  |  |  |
| --- | --- | --- |
| Fid | Name | Type |

CULTIVATES

|  |  |
| --- | --- |
| Fid | Cropid |

NEEDS

|  |  |
| --- | --- |
| Cropid | Feid |

INFLUENCED\_BY

|  |  |  |
| --- | --- | --- |
| Cropid | state | District |

CROP\_REGISTERED\_FOR

|  |  |  |  |
| --- | --- | --- | --- |
| Fid | Cropid | Inid | Doi |

EQUIP\_REGISTERED\_FOR

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Fid | Name | Type | Inid | Doi |

ADVANCED LOGICAL DESIGN

* This is obtained by applying Normalization on the existing relations
* As the values of Inid, Lid, Doi, Dol are NULL initially hence the relation Farmer is decomposed

R: FARMER

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Fid | Name | Age | Address | Income | Doi | Dol | Inid | Lid | District | State |

**Decomposed into**

R1: FARMER

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Fid | Name | Age | Address | Income | District | State |

R2:FARMER\_INS

|  |  |  |
| --- | --- | --- |
| Fid | Doi | Inid |

R3:FARMER\_LOAN

|  |  |  |
| --- | --- | --- |
| Fid | Dol | Lid |

Checking for lossless decomposition :

Fid determines Name,Age,Address,Income,District,State .

Fid determines Doi,Inid

Fid determines Dol,Lid

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Fid | Name | Age | Address | Income | Doi | Dol | Inid | Lid | District | State |
| R1 | a | a | A | a | a | a | a | a |  | a | a |
| R2 | a | a | A | a | a | a | a | a |  | a | a |
| R3 | a | a | A | a | a | a | a | a | a | a | a |

Since last row is full of a’s the decomposition is lossless.

* As soil is a mutivalued attribute hence the below stated relation is decomposed.

R: LOCATION

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| District | State | Weather | Soil | Off\_address |

**Decomposed into**

R1: LOCATION

|  |  |  |  |
| --- | --- | --- | --- |
| District | State | Weather | Off\_address |

R2:LOCATION\_SOIL

|  |  |  |
| --- | --- | --- |
| District | State | Soil |

Here the tuples are repeated for different soil types for the same district and state making the soil attribute as single valued

Checking for lossless decomposition:

District, State determines Weather, Off\_address

District, State determines soil

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | District | State | Weather | Soil | Off\_address |
| R1 | a | a | a |  | a |
| R2 | a | a | a | a | a |

Since last row is full of a’s the decomposition is lossless.

* As all the attributes of all the relations listed above are single valued hence all the relations are in 1 NF.
* All relations have only one attribute as the key except Location and Farm\_equipment.

Even in these 2 relations there are no partial dependencies

Hence all satisfy the 2NF.

* No transitive dependencies in any relation hence all satisfy the 3NF.

Physical Design

A] Assumptions

i) Maximum number of tuples per relation:

1. FARMER 10,000
2. FARMER\_INS 6,000
3. FARMER\_LOAN 6,000
4. CROP 500
5. FERTILIZER 200
6. LOCATION 300
7. LOCATION\_SOIL 900
8. LOAN 50
9. INSURANCE 50
10. FARM\_EQUIPMENT 100
11. UTILIZES 3,000
12. CULTIVATES 6,000
13. NEEDS 1,000
14. INFLUENCED\_BY 1,000
15. CROP\_REGISTERED\_FOR 8,000
16. EQUIP\_REGISTERED\_FOR 5,000

ii) Disk parameters

Disk block size : 1024B

Average seek time : 20ms

Average latency time : 9.5ms

Inter block gap size : 106B

Block transfer time (btt) : 0.5ms

Block pointer size : 5B

B] Storage requirements

Unspanned record blocking of relations is used and is justified while calculating storage requirements of each relation

C] Access methods:

# FARMER

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Fid | Name | Age | Address | Income | District | State | Total |
| 4B | 20B | 4B | 20B | 8B | 10B | 10B | 76B |

**General calculations:**

Blocking factor, ⎣1024/76⎦ =**13**

No. of file blocks required = ⎡10,000/13⎤ = **770**

**Storage Requirements:**

* By choosing “*unspanned record blocking*” the no. of bytes wasted/block =

1024-76\*13 = **36** Bytes

* This makes each record effectively use 1024/13 = 78.77 Bytes of storage. So the overhead is 2.77 Bytes/record. If we use the spanned record blocking we can save this huge loss. Hence the number of blocks required

= ⎡ (10,000\*76)/1024⎤=**743**

* In addition IBG (Inter Block Gap) uses up to 106 bytes/block.
* Total storage required for Farmer file on disk = (Block size + Inter-Block size)\* 743 = (1024+106)\*743) = **820KB**

**Access Method**

1. Since the number of File blocks required is 743, if we use the binary search on this data file, it would need approx.

[log2 743] = **10** block accesses.

1. If we use **primary indexing**,

* The length of each entry in index file is 4 bytes (for key field) + 5 bytes for block pointer = **9** Bytes.
* Index blocking factor = ⎣1024/9⎦ = **113**
* The total no. of entries in index file = no. of blocks in Farmer file = **743**
* Number of blocks needed for index file = ⎡/113⎤ = 7 Blocks.
* To perform binary search on index file we would need [log2 7] =

**3** block accesses.

* To Search for a record we need, 3+2 (to access the data file block containing record) = **5** block accesses
* Additional size required for primary indexing(with interblock gap) is **7.72**KB
* Additional space per record is 7.72\*1024/10000 = **0.79**Bytes / record

1. If we consider **Secondary indexing**

* To construct secondary index on Fid, the size of index entry is 4 bytes + 5 bytes (for block printer) = **9** bytes.
* Blocking factor is ⎣1024/9⎦ = **113** entries/block.
* Total no. of index entries = **10000**.
* No. of blocks needed for index file = ⎡10000/113⎤ = **89** blocks.
* A binary search on this secondary index needs [log2 89] = **7** block accesses.
* To search for a record we need, 7+2 (to access the data file block containing record) = **9** block accesses
* Additional size required for secondary indexing(with interblock gap) is **98.21**KB
* Additional space per record is 98.21\*1024/10000 = **10.06**Bytes /record

1. If we convert secondary index into **multilevel secondary index**,

* Index blocking factor = **113** entries/block.
* Number of 1st level blocks(got from secondary indexing) is 89.
* The no. of 2nd level blocks are ⎡89/113⎤ = 1 block.
* Total blocks required for this is 1+89=**90**.
* Additional size required for secondary indexing with multiplexing(with interblock gap) is **99.32**KB
* No. of block access to access the record is 2+2=**4**.
* So, the additional over head for each record in data file is (99.32\*1024)/10000=**10.17** Bytes/record.

1. Consider the case of **B trees**

* We know, search field (Fid) is 4 bytes long, block pointer = 5 Bytes and Disk block size = 1024 Bytes.

So the order of each node = (p\*5) + (p-1)\*4+ (p-1)\*5<1024

Which leads to p=**73**(consider maximum)

* We can start from root and see how many values and pointers exist in the average at each subsequent level.

Root 1 nodes 72 entries 73 pointers

Level 1 73 nodes 5256entries 5329pointers

Level 2 5329nodes 383688entries 389017pointers

* No. of blocks needed at level ‘n’ , b1 =⎡10000/72⎤= 139
* No. of blocks needed at level n-1, b2 =⎡139/72⎤=2 blocks
* No. of blocks needed at level n-2, b3 =⎡2/72⎤=1 blocks
* No. Of levels in B tree is 3 (n=2).
* No. of access required here is 3+2=**5**.
* No. of blocks needed for B tree = 142
* Additional space for B tree = (139+2+1)KB = **142**KB
* Each record in the file is now effectively using up additional space = (142\*1024)/10000 = **14.54** Bytes/record.

1. Consider the case of **B+ trees:**

* Order of B+ trees in our case is

P\*5+ (p-1)\*4<=1024

Which leads to p=114(maximum value)

Root 1 node 113entries 114pointers

Level1 114nodes 12882 entries 12996pointers

* No.of blocks needed at leaf node (level n) ,b1=⎡10000/113⎤= **89**
* No. Of blocks needed at level n-1 B2=⎡/113⎤=**1**
* No. Of levels in B+ tree is 2 (n=1).
* No. of access required here is 2+2=**4**.
* Total no. of blocks required=89+1=**90**
* Additional space for B+ tree = (89+1)KB = **90**KB
* Additional space for each record is= (90\*1024)/10000 = **9.22** bytes /rec

**Comparing storage space needed, no. of blocks accessed by various file organization for FARMER file:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Total No. of blocks** | **Additional**  **Space/record** | **No. of disk block accesses** |
| Effect of blocking of records with binary search (unspanned) | 743 | 3 bytes | 10 |
| Primary indexing (without multilevel) | 743 +7(index blocks) = 750 | 0.79 bytes | 5 |
| Secondary index  (without multilevel) | 743+ 89(index blocks) = 832 | 10.06 bytes | 9 |
| Secondary index | 743+ 90(index blocks) = 833 | 10.17 bytes | 4 |
| B trees | 743+ 142(B tree blocks) = 885 | 14.54 bytes | 5 |
| B+ trees | 743 + 90 (B+ tree blocks)=833 | 9.22 bytes | 4 |

So we will be using B+ trees for Farmer relation since it has the least no. of disk block accesses with least space overhead per record.

The total storage space required = 820KB+90KB = **910KB**

# FARMER\_INS

|  |  |  |  |
| --- | --- | --- | --- |
| Fid | Doi | Inid | Total |
| 4B | 8B | 4B | 16B |

**General calculations:**

Blocking factor, ⎣1024/16⎦ =**64**

No. of file blocks required = ⎡6,000/64⎤ = **94**

**Storage Requirements:**

* By choosing “*unspanned record blocking*” the no. of bytes wasted/block =

1024-64\*16 = **0** Bytes

* This makes each record effectively use 1024/13 = 16 Bytes of storage, hence causing no overhead. [If we use the spanned record blocking it will only create an extra overhead of accessing extra blocks to get a record].
* In addition IBG (Inter Block Gap) uses up to 106 bytes/block.
* So the total storage required for Farmer\_INS file on disk = (Block size + Inter-Block size)\* 94 = (1024+106)\*94 = **103.73**KB

**Access Method**

1. Since the number of File blocks required is 94, if we use the binary search on this data file, it would need approx.

[log2 94] = **7** block accesses.

1. If we use **primary indexing**,

* The length of each entry in index file is 4 bytes (for key field) + 5 bytes for block pointer = **9** Bytes.
* Index blocking factor = ⎣1024/9⎦ = **113**
* The total no. of entries in index file = no. of blocks in Farmer\_INS file = **94**
* Number of blocks needed for index file = ⎡/113⎤ = 1 Block.
* To search for a record we need, 1+1 (to access the data file block containing record) = **2** block accesses
* Additional size required for primary indexing (with interblock gap) is **1.1KB**
* Additional space per record is 1.1\*1024/6000 = **0.19**Bytes / record.

There is no need to go for primary multilevel indexing as there is only one primary index block.

1. If we consider **Secondary indexing**

* To construct secondary index on Fid, the size of index entry is 4 bytes + 5 bytes (for block printer) = **9** bytes.
* Blocking factor is ⎣1024/9⎦ = **113** entries/block.
* Total no. of index entries = **6000**.
* No. of blocks needed for index file = ⎡6000/113⎤ = **54** blocks.
* A binary search on this secondary index needs [log2 54] = **6** block accesses.
* So to search for a record we need, 6+1 (to access the data file block containing record) = **7** block accesses
* Additional size required for secondary indexing (with interblock gap) is **59.6**KB
* Additional space per record is 59.6\*1024/6000 = **10.17**Bytes /record

1. If we convert secondary index into **multilevel secondary index**,

* We know index blocking factor = **113** entries/block.
* Therefore the no. of 1st level blocks is 54.
* The no. of 2nd level blocks are ⎡54/113⎤ = 1 block.
* Total blocks required for this is 1+54=**55**.
* So, disk space required (with interblock gap) is **60.7**KB.
* No. of block access to access the record is 2+1=**3**.
* So, the additional over head for each record in data file is (55\*1024)/6000=**10.36** Bytes/record.

1. Consider the case of **B trees**

* We know, search field (Fid) is 4 bytes long, block pointer = 5 Bytes and Disk block size = 1024 Bytes.

So the order of each node = (p\*5) + (p-1)\*4+ (p-1)\*5<1024

Which leads to p=**73**(consider maximum)

* We can start from root and see how many values and pointers exist in the average at each subsequent level.

Root 1 nodes 72 entries 73 pointers

Level 1 73 nodes 5256entries 5329pointers

Level 2 5329nodes 383688entries 389017pointers

* No. of blocks needed at level ‘n’ , b1 =⎡6000/72⎤= 84
* No. of blocks needed at level n-1, b2 =⎡84/72⎤=2 blocks
* No. of blocks needed at level n-2, b3 =⎡2/72⎤=1 blocks
* No. Of levels in B tree is 3 (n=2).
* No. of access required here is 3+1=**4**.
* No. of blocks needed for B tree = 84+2+1 = 87
* Additional space for B tree = (84+2+1)KB = **87**KB
* Each record in the file is now effectively using up additional space = (87\*1024)/6000 = **14.84** Bytes/record.

1. Consider the case of **B+ trees:**

* Order of B+ trees in our case is

P\*5+ (p-1)\*4<=1024

Which leads to p=114(maximum value)

Root 1 node 113entries 114pointers

Level1 114nodes 12882 entries 12996pointers

* No.of blocks needed at leaf node (level n) ,b1=⎡6000/113⎤= **54**
* No. Of blocks needed at level n-1 B2=⎡/113⎤=**1**
* No. Of levels in B+ tree is 2 (n=1).
* No. of access required here is 2+1=**3**.
* Total no. of blocks required=54+1=**55**
* Additional space for B+ tree = (54+1)KB = **55**KB
* Additional space for each record is= (55\*1024)/6000 = **9.39** bytes /rec

**Comparing storage space needed, no. of blocks accessed by various file organization for FARMER\_INS file:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Total No. of blocks** | **Additional**  **Space/record** | **No. of disk block accesses** |
| Effect of blocking of records with binary search (unspanned) | 94 | 0.0 bytes | 7 |
| Primary indexing (without multilevel) | 94 +1(index blocks) = 95 | 0.19 bytes | 2 |
| Secondary index  (without multilevel) | 94 + 54(index blocks) = 148 | 10.17 bytes | 7 |
| Secondary index | 94 + 55(index blocks) = 149 | 10.36 bytes | 3 |
| B trees | 94+87(B tree blocks) = 181 | 14.84 bytes | 4 |
| B+ trees | 94 + 55 (B+ tree blocks)=149 | 9.39 bytes | 3 |

So we will be using B+ trees for Farmer\_INS relation since it has the least no. of disk block accesses with least space overhead per record.

The total storage space required=103.73KB+55KB = **158.73KB**

# FARMER\_LOAN

|  |  |  |  |
| --- | --- | --- | --- |
| Fid | Dol | Lid | Total |
| 4B | 8B | 4B | 16B |

**General calculations:**

Blocking factor, ⎣1024/16⎦ =**64**

No. of file blocks required = ⎡6,000/64⎤ = **94**

**Storage Requirements:**

* By choosing “*unspanned record blocking*” the no. of bytes wasted/block =

1024-64\*16 = **0** Bytes

* This makes each record effectively use 1024/13 = 16 Bytes of storage, hence causing no overhead. [If we use the spanned record blocking it will only create an extra overhead of accessing extra blocks to get a record].
* In addition IBG (Inter Block Gap) uses up to 106 bytes/block.
* So the total storage required for Farmer\_Loan file on disk = (Block size + Inter-Block size)\* 94 = (1024+106)\*94 = **103.73**KB

**Access Method**

1. Since the number of File blocks required is 94, if we use the binary search on this data file, it would need approx.

[log2 94] = **7** block accesses.

1. If we use **primary indexing**,

* The length of each entry in index file is 4 bytes (for key field) + 5 bytes for block pointer = **9** Bytes.
* Index blocking factor = ⎣1024/9⎦ = **113**
* The total no. of entries in index file = no. of blocks in Farmer\_Loan file = **94**
* Number of blocks needed for index file = ⎡/113⎤ = 1 Block.
* To search for a record we need, 1+1 (to access the data file block containing record) = **2** block accesses
* Additional size required for primary indexing (with interblock gap) is **1.1KB**
* Additional space per record is 1.1\*1024/6000 = **0.19**Bytes / record.

There is no need to go for primary multilevel indexing as there is only one primary index block.

1. If we consider **Secondary indexing**

* To construct secondary index on Fid, the size of index entry is 4 bytes + 5 bytes (for block printer) = **9** bytes.
* Blocking factor is ⎣1024/9⎦ = **113** entries/block.
* Total no. of index entries = **6000**.
* No. of blocks needed for index file = ⎡6000/113⎤ = **54** blocks.
* A binary search on this secondary index needs [log2 54] = **6** block accesses.
* So to search for a record we need, 6+1 (to access the data file block containing record) = **7** block accesses
* Additional size required for secondary indexing (with interblock gap) is **59.6**KB
* Additional space per record is 59.6\*1024/6000 = **10.17**Bytes /record

1. If we convert secondary index into **multilevel secondary index**,

* We know index blocking factor = **113** entries/block.
* Therefore the no. of 1st level blocks is 54.
* The no. of 2nd level blocks are ⎡54/113⎤ = 1 block.
* Total blocks required for this is 1+54=**55**.
* So, disk space required (with interblock gap) is **60.7**KB.
* No. of block access to access the record is 2+1=**3**.
* So, the additional over head for each record in data file is (55\*1024)/6000=**10.36** Bytes/record.

1. Consider the case of **B trees**

* We know, search field (Fid) is 4 bytes long, block pointer = 5 Bytes and Disk block size = 1024 Bytes.

So the order of each node = (p\*5) + (p-1)\*4+ (p-1)\*5<1024

Which leads to p=**73**(consider maximum)

* We can start from root and see how many values and pointers exist in the average at each subsequent level.

Root 1 nodes 72 entries 73 pointers

Level 1 73 nodes 5256entries 5329pointers

Level 2 5329nodes 383688entries 389017pointers

* No. of blocks needed at level ‘n’ , b1 =⎡6000/72⎤= 84
* No. of blocks needed at level n-1, b2 =⎡84/72⎤=2 blocks
* No. of blocks needed at level n-2, b3 =⎡2/72⎤=1 blocks
* No. Of levels in B tree is 3 (n=2).
* No. of access required here is 3+1=**4**.
* No. of blocks needed for B tree = 84+2+1 = 87
* Additional space for B tree = (84+2+1)KB = **87**KB
* Each record in the file is now effectively using up additional space = (87\*1024)/6000 = **14.84** Bytes/record.

1. Consider the case of **B+ trees:**

* Order of B+ trees in our case is

P\*5+ (p-1)\*4<=1024

Which leads to p=114(maximum value)

Root 1 node 113entries 114pointers

Level1 114nodes 12882 entries 12996pointers

* No.of blocks needed at leaf node (level n) ,b1=⎡6000/113⎤= **54**
* No. Of blocks needed at level n-1 B2=⎡/113⎤=**1**
* No. Of levels in B+ tree is 2 (n=1).
* No. of access required here is 2+1=**3**.
* Total no. of blocks required=54+1=**55**
* Additional space for B+ tree = (54+1)KB = **55**KB
* Additional space for each record is= (55\*1024)/6000 = **9.39** bytes /rec

**Comparing storage space needed, no. of blocks accessed by various file organization for FARMER\_LOAN file:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Total No. of blocks** | **Additional**  **Space/record** | **No. of disk block accesses** |
| Effect of blocking of records with binary search (unspanned) | 94 | 0.0 bytes | 7 |
| Primary indexing (without multilevel) | 94 +1(index blocks) = 95 | 0.19 bytes | 2 |
| Secondary index  (without multilevel) | 94 + 54(index blocks) = 148 | 10.17 bytes | 7 |
| Secondary index | 94 + 55(index blocks) = 149 | 10.36 bytes | 3 |
| B trees | 94+87(B tree blocks) = 181 | 14.84 bytes | 4 |
| B+ trees | 94 + 55 (B+ tree blocks)=149 | 9.39 bytes | 3 |

So we will be using B+ trees for Farmer\_Loan relation since it has the least no. of disk block accesses with least space overhead per record.

The total storage space required=103.73KB+55KB =**158.73KB**

# CROP

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Cropid | Type | Name | Soil | Disease | Water | Price | Seeds\_price | Total |
| 4B | 10B | 10B | 10B | 10B | 8B | 8B | 8B | 68B |

**General calculations:**

Blocking factor, ⎣1024/68⎦ =**15**

No. of file blocks required = ⎡500/15⎤ = **34**

**Storage Requirements:**

* By choosing “*unspanned record blocking*” the no. of bytes wasted/block =

1024-68\*15 = **4** Bytes

* This makes each record effectively use 1024/15 = 16.27 Bytes of storage. So the overhead is only 3 Bytes/record. [If we use the spanned record blocking we can save this but it leads to complexity in accessing blocks, like sometimes we need to access 2 blocks to get a record].
* In addition IBG (Inter Block Gap) uses up to 106 bytes/block.
* So the total storage required for Crop file on disk = (Block size + Inter-Block size)\* 34 = (1024+106)\*34 = **37.51**KB

**Access Method**

1. Since the number of File blocks required is 34, if we use the binary search on this data file, it would need approx.

[log2 34] = **6** block accesses.

1. If we use **primary indexing**,

* The length of each entry in index file is 4 bytes (for key field) + 5 bytes for block pointer = **9** Bytes.
* Index blocking factor = ⎣1024/9⎦ = **113**
* The total no. of entries in index file = no. of blocks in Crop file = **34**
* Number of blocks needed for index file = ⎡/113⎤ = 1 Block.
* So to search for a record we need, 1+1 (to access the data file block containing record) = **2** block accesses
* Additional size required for primary indexing(with interblock gap) is **1.1**KB
* Additional space per record is 1.1\*1024/500 = **2.25**Bytes / record.

There is no need to go for primary multilevel indexing as there is only one primary index block.

1. If we consider **Secondary indexing**

* To construct secondary index on Cropid, the size of index entry is 4 bytes + 5 bytes (for block printer) = **9** bytes.
* Blocking factor is ⎣1024/9⎦ = **113** entries/block.
* Total no. of index entries = **500**.
* No. of blocks needed for index file = ⎡500/113⎤ = **5** blocks.
* A binary search on this secondary index needs [log2 5] = **3** block accesses.
* So to search for a record we need, 3+1 (to access the data file block containing record) = **4** block accesses
* Additional size required for secondary indexing(with interblock gap) is **5.52**KB
* Additional space per record is 5.52\*1024/500 = **11.30**Bytes /record

1. If we convert secondary index into **multilevel secondary index**,

* We know index blocking factor = **113** entries/block.
* Therefore the no. of 1st level blocks is 5.
* The no. of 2nd level blocks are ⎡5/113⎤ = 1 block.
* Total blocks required for this is 1+5=**6**.
* So, disk space required(with interblock gap) is 6.62KB.
* No. of block access to access the record is 2+1=**3**.
* So, the additional over head for each record in data file is (6\*1024)/500= **13.56** Bytes/record.

1. Consider the case of **B trees**

* We know, search field (Fid) is 4 bytes long, block pointer = 5 Bytes and Disk block size = 1024 Bytes.

So the order of each node = (p\*5) + (p-1)\*4+ (p-1)\*5<1024

Which leads to p=**73**(consider maximum)

* We can start from root and see how many values and pointers exist in the average at each subsequent level.

Root 1 nodes 72 entries 73 pointers

Level 1 73 nodes 5256entries 5329pointers

Level 2 5329nodes 383688entries 389017pointers

* No. of blocks needed at level ‘n’ , b1 =⎡500/72⎤= 7
* No. of blocks needed at level n-1, b2 =⎡7/72⎤=1 blocks
* No. Of levels in B tree is 2 (n=1).
* No. of access required here is 2+1=**3**.
* No. of blocks needed for B tree = 7+1 = 8
* Additional space for B tree = (7+1)KB = **8**KB
* Each record in the file is now effectively using up additional space = (8\*1024)/500 = **16.38** Bytes/record.

1. Consider the case of **B+ trees:**

* Order of B+ trees in our case is

P\*5+ (p-1)\*4<=1024

Which leads to p=114(maximum value)

Root 1 node 113entries 114pointers

Level1 114nodes 12882 entries 12996pointers

* No.of blocks needed at leaf node (level n) ,b1=⎡500/113⎤= **5**
* No. Of blocks needed at level n-1 B2=⎡/113⎤=**1**
* No. Of levels in B+ tree is 2 (n=1).
* No. of access required here is 2+1=**3**.
* Total no. of blocks required=5+1=**6**
* Additional space for B+ tree = (5+1)KB = **6**KB
* Additional space for each record is= (6\*1024)/500 = **12.29** bytes /rec

**Comparing storage space needed, no. of blocks accessed by various file organization for Crop file:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Total No. of blocks** | **Additional**  **Space/record** | **No. of disk block accesses** |
| Effect of blocking of records with binary search (unspanned) | 34 | 0.27 bytes | 6 |
| Primary indexing (without multilevel) | 34 +1(index blocks) = 35 | 2.25 bytes | 2 |
| Secondary index  (without multilevel) | 34 + 5(index blocks) = 39 | 11.30 bytes | 4 |
| Secondary index | 34 + 6(index blocks) = 40 | 13.65 bytes | 3 |
| B trees | 34+8(B tree blocks) = 42 | 16.38 bytes | 3 |
| B+ trees | 34 + 6 (B+ tree blocks)=40 | 12.29 bytes | 3 |

So we will be using primary indexing for Crop relation since it has the least no. of disk block accesses.

The total storage space required=37.51KB+1KB(for primary index block) =**38.51KB**

# Fertilizer

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Feid | Name | Type | Distributor | Price | Total |
| 4B | 10B | 10B | 10B | 8B | 42B |

**General calculations:**

Blocking factor, ⎣1024/42⎦ =**24**

No. of file blocks required = ⎡200/24⎤ = **9**

**Storage Requirements:**

* By choosing “*unspanned record blocking*” the no. of bytes wasted/block =

1024-24\*42 = **16** Bytes

* This makes each record effectively use 1024/24 = 42.67 Bytes of storage. So the overhead is only 0.67 Bytes/record. [If we use the spanned record blocking we can save this but it leads to complexity in accessing blocks, like sometimes we need to access 2 blocks to get a record].
* In addition IBG (Inter Block Gap) uses up to 106 bytes/block.
* Total storage required for Fertilizer file on disk = (Block size + Inter-Block size)\* 9 = (1024+106)\*9 = **9.92**KB

**Access Method**

1. Since the number of File blocks required is 9, if we use the binary search on this data file, it would need approx.

[log2 9] = **4** block accesses.

1. If we use **primary indexing**,

* The length of each entry in index file is 4 bytes (for key field) + 5 bytes for block pointer = **9** Bytes.
* Index blocking factor = ⎣1024/9⎦ = **113**
* The total no. of entries in index file = no. of blocks in Fertilizer file = **9**
* Number of blocks needed for index file = ⎡/113⎤ = 1 Blocks.
* To Search for a record we need, 3+1 (to access the data file block containing record) = **4** block accesses
* Additional size required for primary indexing(with interblock gap) is **1.1**KB
* Additional space per record is (1.1\*1024)/200 = **5.63**Bytes / record

1. If we consider **Secondary indexing**

* To construct secondary index on Feid, the size of index entry is 4 bytes + 5 bytes (for block printer) = **9** bytes.
* Blocking factor is ⎣1024/9⎦ = **113** entries/block.
* Total no. of index entries = **200**.
* No. of blocks needed for index file = ⎡200/113⎤ = **2** blocks.
* A binary search on this secondary index needs [log2 2] = **1** block accesses.
* To search for a record we need, 1+1 (to access the data file block containing record) = **2** block accesses
* Additional size required for secondary indexing(with interblock gap) is **2.2**KB
* Additional space per record is 2.2\*1024/200 = **11.26**Bytes /record

Since the number of block in secondary and primary indexing is very low(2 and 1 respectively), we are not considering multilevel indexing

1. Consider the case of **B trees**

* We know, search field (Fid) is 4 bytes long, block pointer = 5 Bytes and Disk block size = 1024 Bytes.

So the order of each node = (p\*5) + (p-1)\*4+ (p-1)\*5<1024

Which leads to p=**73**(consider maximum)

* We can start from root and see how many values and pointers exist in the average at each subsequent level.

Root 1 nodes 72 entries 73 pointers

Level 1 73 nodes 5256entries 5329pointers

Level 2 5329nodes 383688entries 389017pointers

* No. of blocks needed at level ‘n’ , b1 =⎡200/72⎤= 3
* No. of blocks needed at level n-1, b2 =⎡3/72⎤=1 blocks
* No. Of levels in B tree is 2 (n=1).
* No. of access required here is 2+1=**3**.
* No. of blocks needed for B tree = 4
* Additional space for B tree = (3+1)KB = **4**KB
* Each record in the file is now effectively using up additional space = (4\*1024)/200 = **20.48** Bytes/record.

1. Consider the case of **B+ trees:**

* Order of B+ trees in our case is

P\*5+ (p-1)\*4<=1024

Which leads to p=114(maximum value)

Root 1 node 113entries 114pointers

Level1 114nodes 12882 entries 12996pointers

* No.of blocks needed at leaf node (level n) ,b1=⎡200/113⎤= **2**
* No. Of blocks needed at level n-1 B2=⎡/113⎤=**1**
* No. Of levels in B+ tree is 2 (n=1).
* No. of access required here is 2+1=**3**.
* Total no. of blocks required=2+1=**3**
* Additional space for B+ tree = (2+1)KB = **3**KB
* Additional space for each record is= (3\*1024)/200 = **15.36** bytes /rec

**Comparing storage space needed, no. of blocks accessed by various file organization for Fertilizer file:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **No. of blocks** | **Additional**  **Space/record** | **No. of disk block accesses** |
| Effect of blocking of records with binary search (unspanned) | 9 | 0.67 bytes | 4 |
| Primary indexing (without multilevel) | 9 + 1(index file) = 10 | 5.36 bytes | 2 |
| Secondary index  (without multilevel) | 9 + 2(index file) = 11 | 11.26 bytes | 2 |
| B trees | 9 + 4(B tree)= 13 | 20.48 bytes | 3 |
| B+ trees | 9 + 3(B+ tree) = 12 | 15.36 bytes | 3 |

So we will be using Secondary indexing for Fertilizer relation since it has the least no. of disk block accesses and data file need not be ordered when compared to primary indexing.

The total storage space required=9.92KB+2.2KB =**12.13KB**

# LOCATION

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| District | State | Weather | Off\_address | Total |
| 10B | 10B | 10B | 10B | 40B |

**General calculations:**

Blocking factor, ⎣1024/40⎦ =**25**

No. of file blocks required = ⎡300/25⎤ = **12**

**Storage Requirements:**

* By choosing “*unspanned record blocking*” the no. of bytes wasted/block =

1024-40\*25 = **24** Bytes

* This makes each record effectively use 1024/25 = 40.96 Bytes of storage. So the overhead is only 0.96 Bytes/record. [If we use the spanned record blocking we can save this but it leads to complexity in accessing blocks, like sometimes we need to access 2 blocks to get a record].
* In addition IBG (Inter Block Gap) uses up to 106 bytes/block.
* So the total storage required for Location file on disk = (Block size + Inter-Block size)\* 12 = (1024+106)\*12 = **13.24**KB

**Access Method**

1. Since the number of File blocks required is 12, if we use the binary search on this data file, it would need approx.

[log2 12] = **4** block accesses.

1. If we consider **Secondary indexing(based on any one attribute)**

* To construct secondary index on District (or state), the size of index entry is 10 bytes + 5 bytes (for block printer) = **15** bytes.
* Blocking factor is ⎣1024/15⎦ = **68** entries/block.
* Total no. of index entries = **300**.
* No. of blocks needed for index file = ⎡300/68⎤ = **5** blocks.
* A binary search on this secondary index needs [log2 5] = **3** block accesses.
* So to search for a record we need, 3+1 (to access the data file block containing record) = **4** block accesses
* Additional size required for secondary indexing(with interblock gap) is **5.52**KB
* Additional space per record is 5.52\*1024/300 = **18.84**Bytes /record

1. If we convert secondary index into **multilevel secondary index**,

* We know index blocking factor = **68** entries/block.
* Therefore the no. of 1st level blocks is 5.
* The no. of 2nd level blocks are ⎡5/68⎤ = 1 block.
* Total blocks required for this is 1+5=**6**.
* So, disk space required (with interblock gap) is 6.62KB.
* No. of block access to access the record is 2+1=**3**.
* So, the additional over head for each record in data file is (6.62\*1024)/300= **22.6** Bytes/record.

1. Consider the case of **B trees**

* We know, search field (Fid) is 4 bytes long, block pointer = 5 Bytes and Disk block size = 1024 Bytes.

So the order of each node = (p\*5) + (p-1)\*10+ (p-1)\*5<1024

Which leads to p=**51**(consider maximum)

* We can start from root and see how many values and pointers exist in the average at each subsequent level.

Root 1 nodes 50 entries 51 pointers

Level 1 51 nodes 2550entries 2601pointers

Level 2 2601nodes 130050entries 132651pointers

* No. of blocks needed at level ‘n’ , b1 =⎡300/51⎤= 6
* No. of blocks needed at level n-1, b2 =⎡51⎤=1 blocks
* No. Of levels in B tree is 2 (n=1).
* No. of access required here is 2+1=**3**.
* No. of blocks needed for B tree = 6+1 = 6
* Additional space for B tree = (6+1)KB = **7**KB
* Each record in the file is now effectively using up additional space = (7\*1024)/300 = **23.9** Bytes/record.

1. Consider the case of **B+ trees:**

* Order of B+ trees in our case is

P\*5+ (p-1)\*10<=1024

Which leads to p=68(maximum value)

Root 1 node 67entries 68pointers

Level1 68nodes 4556 entries 4624pointers

* No.of blocks needed at leaf node (level n) ,b1=⎡300/68⎤= **4**
* No. Of blocks needed at level n-1 B2=⎡⎤=**1**
* No. Of levels in B+ tree is 2 (n=1).
* No. of access required here is 2+1=**3**.
* Total no. of blocks required=4+1=**5**
* Additional space for B+ tree = (4+1)KB = **5**KB
* Additional space for each record is= (5\*1024)/300 = **17.07** bytes /rec

**Comparing storage space needed, no. of blocks accessed by various file organization for Location file:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Total No. of blocks** | **Additional**  **Space/record** | **No. of disk block accesses** |
| Effect of blocking of records with binary search (unspanned) | 12 | 0.96 bytes | 4 |
| Secondary index  (without multilevel) | 12 + 5(index blocks) = 17 | 18.84 bytes | 4 |
| Secondary index | 12 + 6(index blocks) = 18 | 22.6 bytes | 3 |
| B trees | 12+6(B tree blocks) = 18 | 23.9 bytes | 3 |
| B+ trees | 12 + 4 (B+ tree blocks)=16 | 17.07 bytes | 3 |

So we will be using B+ trees for Location relation since it has the least no. of disk block along with lowest additional space requirement.

The total storage space required=13.24KB+4KB (for B+ tree) =**17.24KB**

# LOCATION\_SOIL

|  |  |  |  |
| --- | --- | --- | --- |
| District | State | Soil | Total |
| 10B | 10B | 10B | 30B |

**General calculations:**

Blocking factor, ⎣1024/30⎦ =**34**

No. of file blocks required = ⎡900/34⎤ = **27**

**Storage Requirements:**

* By choosing “*unspanned record blocking*” the no. of bytes wasted/block =

1024-30\*34 = **4** Bytes

* This makes each record effectively use 1024/34 = 40.12 Bytes of storage. So the overhead is only 0.12 Bytes/record. [If we use the spanned record blocking we can save this but it leads to complexity in accessing blocks, like sometimes we need to access 2 blocks to get a record].
* In addition IBG (Inter Block Gap) uses up to 106 bytes/block.
* Total storage required for Location\_soil file on disk = (Block size + Inter-Block size)\* 27 = (1024+106)\*27= **29.79**KB

**Access Method**

1. Since the number of File blocks required is 12, if we use the binary search on this data file, it would need approx.

[log2 27] = **5** block accesses.

1. If we consider **Secondary indexing(based on any one attribute)**

* To construct secondary index on District (or state), the size of index entry is 10 bytes + 5 bytes (for block printer) = **15** bytes.
* Blocking factor is ⎣1024/15⎦ = **68** entries/block.
* Total no. of index entries = **900**.
* No. of blocks needed for index file = ⎡900/68⎤ = **14** blocks.
* A binary search on this secondary index needs [log2 14] = **4** block accesses.
* So to search for a record we need, 4+1 (to access the data file block containing record) = **5** block accesses
* Additional size required for secondary indexing(with interblock gap) is **15.5**KB
* Additional space per record is 15.5\*1024/900 = **17.64**Bytes /record

1. If we convert secondary index into **multilevel secondary index**,

* We know index blocking factor = **68** entries/block.
* Therefore the no. of 1st level blocks is 14.
* The no. of 2nd level blocks are ⎡14/68⎤ = 1 block.
* Total blocks required for this is 1+14=**15**.
* So, disk space required is 16.6KB.
* No. of block access to access the record is 2+1=**3**.
* So, the additional over head for each record in data file is (16.6\*1024)/900= **18.88** Bytes/record.

1. Consider the case of **B trees**

* We know, search field (Fid) is 4 bytes long, block pointer = 5 Bytes and Disk block size = 1024 Bytes.

So the order of each node = (p\*5) + (p-1)\*10+ (p-1)\*5<1024

Which leads to p=**51**(consider maximum)

* We can start from root and see how many values and pointers exist in the average at each subsequent level.

Root 1 nodes 50 entries 51 pointers

Level 1 51 nodes 2550entries 2601pointers

Level 2 2601nodes 130050entries 132651pointers

* No. of blocks needed at level ‘n’ , b1 =⎡900/51⎤= 17
* No. of blocks needed at level n-1, b2 =⎡17/72⎤=1 blocks
* No. Of levels in B tree is 2 (n=1).
* No. of access required here is 2+1=**3**.
* No. of blocks needed for B tree = 17+1 = 18
* Additional space for B tree = (17+1)KB = **18**KB
* Each record in the file is now effectively using up additional space = (18\*1024)/900 = **20.48** Bytes/record.

1. Consider the case of **B+ trees:**

* Order of B+ trees in our case is

P\*5+ (p-1)\*10<=1024

Which leads to p=68(maximum value)

Root 1 node 67entries 68pointers

Level1 68nodes 4556 entries 4624pointers

* No.of blocks needed at leaf node (level n) ,b1=⎡900/68⎤= **13**
* No. Of blocks needed at level n-1 B2=⎡/68⎤=**1**
* No. Of levels in B+ tree is 2 (n=1).
* No. of access required here is 2+1=**3**.
* Total no. of blocks required=13+1=**14**
* Additional space for B+ tree = (13+1)KB = **14**KB
* Additional space for each record is= (14\*1024)/900 = **15.92** bytes /rec

**Comparing storage space needed, no. of blocks accessed by various file organization for Location\_soil file:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Total No. of blocks** | **Additional**  **Space/record** | **No. of disk block accesses** |
| Effect of blocking of records with binary search (unspanned) | 27 | 0.12 bytes | 5 |
| Secondary index  (without multilevel) | 27 + 14(index blocks) = 41 | 17.64 bytes | 5 |
| Secondary index | 27 + 15(index blocks) = 42 | 18.88 bytes | 3 |
| B trees | 27+14(B tree blocks) = 41 | 10.48 bytes | 3 |
| B+ trees | 12 + 9 (B+ tree blocks)=21 | 15.92 bytes | 3 |

So we will be using B+ trees for Location\_soil relation since it has the least no. of disk block along with lowest additional space requirement.

The total storage space required=29.79KB+9KB (for B+ tree) =**38.79KB**

# LOAN

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Lid | Init\_amount | Term | Bank | Income\_status | Rate | Total |
| 4B | 8B | 4B | 10B | 8B | 4B | 38B |

**General calculations:**

Blocking factor, ⎣1024/38⎦ =**26**

No. of file blocks required = ⎡50/26⎤ = **2**

**Storage Requirements:**

* By choosing “*unspanned record blocking*” the no. of bytes wasted/block =

1024-26\*38 = **36** Bytes

* This makes each record effectively use 1024/26 = 39.38 Bytes of storage. So the overhead is only 1.38 Bytes/record. [If we use the spanned record blocking we can save this but it leads to complexity in accessing blocks, like sometimes we need to access 2 blocks to get a record].
* In addition IBG (Inter Block Gap) uses up to 106 bytes/block.
* Total storage required for Loan file on disk = (Block size + Inter-Block size)\* 4 = (1024+106)\*2= 2.21KB

**Access Method**

1. Since the number of File blocks required is 2, if we use the binary search on this data file, it would need approx.

[log2 2] = **1** block accesses.

Since There are only 2 blocks We shall not consider Primary indexing as it provides no benefit but requires a block access for index block

1. If we consider **Secondary indexing**

* To construct secondary index on Lid, the size of index entry is 4 bytes + 5 bytes (for block printer) = **9** bytes.
* Blocking factor is ⎣1024/9⎦ = **113** entries/block.
* Total no. of index entries = **50**.
* No. of blocks needed for index file = ⎡50/113⎤ = **1** blocks.
* A binary search on this secondary index needs [log2 1] = **1** block accesses.
* So to search for a record we need, 1+1 (to access the data file block containing record) = **2** block accesses
* Additional size required for secondary indexing(with interblock gap) is **1.1**KB
* Additional space per record is 1.1\*1024/50 = **22.52**Bytes /record

Since there is only one index block in above case, we are not going for multilevel indexing.

Also since the number of records is very low, we are not considering B and B+ trees.

**Comparing storage space needed, no. of blocks accessed by various file organization for Loan file:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **No. of blocks** | **Additional**  **Space/record** | **No. of disk block accesses** |
| Effect of blocking of records with binary search (unspanned) | 4 | 2 bytes | 2 |
| Secondary index  (without multilevel) | 4 + 1(index block) = 5 | 22.52 bytes | 2 |

So we will be using secondary indexing for Loan relation. Even though Direct binary search on blocks requires lesser number of disk access, it requires ordering of the data file, but secondary indexing does not need the ordering of data file.

The total storage space required=2.21KB+1.1KB =**3.31KB**

# INSURANCE

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Inid | Name | Income\_status | Type | Premium | Expiry | Amt\_claimable | Total |
| 4B | 20B | 8B | 10B | 8B | 4B | 8B | 62B |

**General calculations:**

Blocking factor, ⎣1024/62⎦ =**16**

No. of file blocks required = ⎡50/16⎤ = **4**

**Storage Requirements:**

* By choosing “*unspanned record blocking*” the no. of bytes wasted/block =

1024-16\*62 = **32** Bytes

* This makes each record effectively use 1024/16 = 64 Bytes of storage. So the overhead is only 2 Bytes/record. [If we use the spanned record blocking we can save this but it leads to complexity in accessing blocks, like sometimes we need to access 2 blocks to get a record].
* In addition IBG (Inter Block Gap) uses up to 106 bytes/block.
* Total storage required for Insurance file on disk = (Block size + Inter-Block size)\* 4 = (1024+106)\*4= **4.41**KB

**Access Method**

1. Since the number of File blocks required is 4, if we use the binary search on this data file, it would need approx.

[log2 4] = **2** block accesses.

1. If we use **primary indexing**,

* The length of each entry in index file is 4 bytes (for key field) + 5 bytes for block pointer = **9** Bytes.
* Index blocking factor = ⎣1024/9⎦ = **113**
* The total no. of entries in index file = no. of blocks in Insurance file = **4**
* Number of blocks needed for index file = ⎡/113⎤ = 1 Blocks.
* To perform binary search on index file we would need [log2 1] =

**1** block accesses.

* So to search for a record we need, 1+1 (to access the data file block containing record) = **2** block accesses
* Additional size required for primary indexing(with interblock gap) is **1.1**KB
* Additional space per record is 1.1\*1024/50 = **22.52**Bytes / record

1. If we consider **Secondary indexing**

* To construct secondary index on Inid, the size of index entry is 4 bytes + 5 bytes (for block printer) = **9** bytes.
* Blocking factor is ⎣1024/9⎦ = **113** entries/block.
* Total no. of index entries = **50**.
* No. of blocks needed for index file = ⎡50/113⎤ = **1** blocks.
* A binary search on this secondary index needs [log2 1] = **1** block accesses.
* So to search for a record we need, 1+1 (to access the data file block containing record) = **2** block accesses
* Additional size required for secondary indexing(with interblock gap) is **1.1**KB
* Additional space per record is 1.1\*1024/50 = **22.52**Bytes /record

Since there is only one index block in both the above cases, we are not going for multilevel indexing.

Also since the number of records is very low, we are not considering B and B+ trees.

**Comparing storage space needed, no. of blocks accessed by various file organization for INSURANCE file:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **No. of blocks** | **Additional**  **Space/record** | **No. of disk block accesses** |
| Effect of blocking of records with binary search (unspanned) | 4 | 2 bytes | 2 |
| Primary indexing (without multilevel) | 4 + 1(index block) = 5 | 22.52 bytes | 2 |
| Secondary index  (without multilevel) | 4 + 1(index block) = 5 | 22.52 bytes | 2 |

So we will be using secondary indexing for Insurance relation since the data file need not be ordered for every insert, update/delete operation. The searching takes same number of block access in each case.

The total storage space required = 4.41KB+1KB = **5.41KB**

# FARM\_EQUIPMENT

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Type | Manufacturer | Distributor | Total |
| 20B | 10B | 10B | 10B | 50B |

**General calculations:**

Blocking factor, ⎣1024/50⎦ =**20**

No. of file blocks required = ⎡100/20⎤ = **5**

**Storage Requirements:**

* By choosing “*unspanned record blocking*” the no. of bytes wasted/block =

1024-50\*20 = **24** Bytes

* This makes each record effectively use 1024/20 = 50.48 Bytes of storage. So the overhead is only 0.48 Bytes/record. [If we use the spanned record blocking we can save this but it leads to complexity in accessing blocks, like sometimes we need to access 2 blocks to get a record].
* In addition IBG (Inter Block Gap) uses up to 106 bytes/block.
* Total storage required for Farm\_equipment file on disk = (Block size + Inter-Block size)\* 5 = (1024+106)\*5= **5.65**KB

**Access Method**

1. Since the number of File blocks required is 5, if we use the binary search on this data file, it would need approx.

[log2 5] = **3** block accesses.

1. If we consider **Secondary indexing(based on any one attribute)**

* To construct secondary index on Type (because it takes lesser space), the size of index entry is 10 bytes + 5 bytes (for block printer) = **15** bytes.
* Blocking factor is ⎣1024/15⎦ = **68** entries/block.
* Total no. of index entries = **300**.
* No. of blocks needed for index file = ⎡100/68⎤ = **2** blocks.
* A binary search on this secondary index needs [log2 2] = **1** block accesses.
* So to search for a record we need, 1+1 (to access the data file block containing record) = **2** block accesses
* Additional size required for secondary indexing(with interblock gap) is **2.2**KB
* Additional space per record is 2.2\*1024/100 = **22.52**Bytes /record

Since there are only 2 Index blocks we are not going for multilevel indexing.

Also since the number of records is very low, we are not considering B and B+ trees.

**Comparing storage space needed, no. of blocks accessed by various file organization for Farm\_equipment file:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Total No. of blocks** | **Additional**  **Space/record** | **No. of disk block accesses** |
| Effect of blocking of records with binary search (unspanned) | 5 | 0.58 bytes | 3 |
| Secondary index  (without multilevel) | 5 + 2(index blocks) = 7 | 22.52 bytes | 2 |

So we will be using secondary indexing for Farm\_equipment relation since it has the least no. of disk block accesses.

The total storage space required = 5.65KB+2KB (for secondary index blocks) =**7.65KB**

# UTILIZES

|  |  |  |
| --- | --- | --- |
| Fid | Eqid | Total |
| 4B | 4B | 8B |

**General calculations:**

Blocking factor, ⎣1024/8⎦ =1**28**

No. of file blocks required = ⎡3000/128⎤ = **24**

**Storage Requirements:**

* By choosing “*unspanned record blocking*” the no. of bytes wasted/block =

1024-128\*8 = **0** Bytes

* This makes each record effectively use 1024/13 = 16 Bytes of storage, hence causing no overhead. [If we use the spanned record blocking it will only create an extra overhead of accessing extra blocks to get a record].
* In addition IBG (Inter Block Gap) uses up to 106 bytes/block.
* Total storage required for Utilizes file on disk = (Block size + Inter-Block size)\* 5 = (1024+106)\*24= **26.48**KB

**Access Method**

1. Since the number of File blocks required is 24, if we use the binary search on this data file, it would need approx.

[log2 24] = **5** block accesses.

1. If we consider **Secondary indexing(based on any one attribute)**

* To construct secondary index on Fid (or EqId), the size of index entry is 4 bytes + 5 bytes (for block printer) = **9** bytes.
* Blocking factor is ⎣1024/9⎦ = **113** entries/block.
* Total no. of index entries = **3000**.
* No. of blocks needed for index file = ⎡3000/113⎤ = **27** blocks.
* A binary search on this secondary index needs [log2 27] = **5** block accesses.
* So to search for a record we need, 5+1 (to access the data file block containing record) = **6** block accesses
* Additional size required for secondary indexing(with interblock gap) is **29.8**KB
* Additional space per record is 29.8\*1024/3000 = **10.17**Bytes /record.

1. If we convert secondary index into **multilevel secondary index**,

* Index blocking factor = **113** entries/block.
* Number of 1st level blocks(got from secondary indexing) is 27.
* The no. of 2nd level blocks are ⎡27/113⎤ = 1 block.
* Total blocks required for this is 1+27=**28**.
* Additional size required for secondary indexing with multiplexing(with interblock gap) is **30.9**KB
* No. of block access to access the record is 2+1=**3**.
* So, the additional over head for each record in data file is (30.9\*1024)/3000=**10.55** Bytes/record.

1. Consider the case of **B trees**

* We know, search field (Fid) is 4 bytes long, block pointer = 5 Bytes and Disk block size = 1024 Bytes.

So the order of each node = (p\*5) + (p-1)\*4+ (p-1)\*5<1024

Which leads to p=**73**(consider maximum)

* We can start from root and see how many values and pointers exist in the average at each subsequent level.

Root 1 nodes 72 entries 73 pointers

Level 1 73 nodes 5256entries 5329pointers

Level 2 5329nodes 383688entries 389017pointers

* No. of blocks needed at level ‘n’ , b1 =⎡3000/72⎤= 42
* No. of blocks needed at level n-1, b2 =⎡42/72⎤=1 blocks
* No. Of levels in B tree is 2 (n=1).
* No. of access required here is 2+1=**3**.
* No. of blocks needed for B tree = 43
* Additional space for B tree = (42+1)KB = **43**KB
* Each record in the file is now effectively using up additional space = (43\*1024)/3000 = **14.68** Bytes/record.

1. Consider the case of **B+ trees:**

* Order of B+ trees in our case is

P\*5+ (p-1)\*4<=1024

Which leads to p=114(maximum value)

Root 1 node 113entries 114pointers

Level1 114nodes 12882 entries 12996pointers

* No. of blocks needed at leaf node (level n) ,b1=⎡3000/113⎤= **27**
* No. Of blocks needed at level n-1 B2=⎡/113⎤=**1**
* No. Of levels in B+ tree is 2 (n=1).
* No. of access required here is 2+1=**3**.
* Total no. of blocks required=27+1=**28**
* Additional space for B+ tree = (27+1)KB = **28**KB
* Additional space for each record is= (28\*1024)/3000 = **9.56** bytes /record

**Comparing storage space needed, no. of blocks accessed by various file organization for Utilizes file:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Total No. of blocks** | **Additional**  **Space/record** | **No. of disk block accesses** |
| Effect of blocking of records with binary search (unspanned) | 24 | 0 bytes | 5 |
| Secondary index  (without multilevel) | 24 + 27(index blocks) = 51 | 10.17 bytes | 6 |
| Secondary index | 24 + 28(index blocks) = 52 | 10.55 bytes | 3 |
| B trees | 24 + 43(B tree blocks) = 67 | 14.68 bytes | 3 |
| B+ trees | 24 + 28 (B+ tree blocks)= 52 | 9.56 bytes | 3 |

So we will be using B+ tree indexing for Utilizes relation since it has the least no. of disk block accesses and least additional space/record.

The total storage space required=26.48KB+28KB (for B+ tree) =**54.48KB**

# CULTIVATES

|  |  |  |
| --- | --- | --- |
| Fid | Eqid | Total |
| 4B | 4B | 8B |

**General calculations:**

Blocking factor, ⎣1024/8⎦ =1**28**

No. of file blocks required = ⎡6000/128⎤ = **47**

**Storage Requirements:**

* By choosing “*unspanned record blocking*” the no. of bytes wasted/block =

1024-128\*8 = **0** Bytes

* This makes each record effectively use 1024/13 = 16 Bytes of storage, hence causing no overhead. [If we use the spanned record blocking it will only create an extra overhead of accessing extra blocks to get a record].
* In addition IBG (Inter Block Gap) uses up to 106 bytes/block.
* Total storage required for Cultivates file on disk = (Block size + Inter-Block size)\* 5 = (1024+106)\*47= **51.87**KB

**Access Method**

1. Since the number of File blocks required is 47, if we use the binary search on this data file, it would need approx.

[log2 47] = **6** block accesses.

1. If we consider **Secondary indexing(based on any one attribute)**

* To construct secondary index on Fid (or CropID), the size of index entry is 4 bytes + 5 bytes (for block printer) = **9** bytes.
* Blocking factor is ⎣1024/9⎦ = **113** entries/block.
* Total no. of index entries = **6000**.
* No. of blocks needed for index file = ⎡6000/113⎤ = **54** blocks.
* A binary search on this secondary index needs [log2 54] = **6** block accesses.
* So to search for a record we need, 6+1 (to access the data file block containing record) = **7** block accesses
* Additional size required for secondary indexing(with interblock gap) is **59.58**KB
* Additional space per record is 59.58\*1024/6000 = **10.17**Bytes /record.

1. If we convert secondary index into **multilevel secondary index**,

* Index blocking factor = **113** entries/block.
* Number of 1st level blocks(got from secondary indexing) is 54.
* The no. of 2nd level blocks are ⎡54/113⎤ = 1 block.
* Total blocks required for this is 1+54=**55**.
* Additional size required for secondary indexing with multiplexing(with interblock gap) is **60.68**KB
* No. of block access to access the record is 2+1=**3**.
* So, the additional over head for each record in data file is (60.68\*1024)/3000=**10.55** Bytes/record.

1. Consider the case of **B trees**

* We know, search field (Fid) is 4 bytes long, block pointer = 5 Bytes and Disk block size = 1024 Bytes.

So the order of each node = (p\*5) + (p-1)\*4+ (p-1)\*5<1024

Which leads to p=**73**(consider maximum)

* We can start from root and see how many values and pointers exist in the average at each subsequent level.

Root 1 nodes 72 entries 73 pointers

Level 1 73 nodes 5256entries 5329pointers

Level 2 5329nodes 383688entries 389017pointers

* No. of blocks needed at level ‘n’ , b1 =⎡6000/72⎤= 84
* No. of blocks needed at level n-1, b2 =⎡84/72⎤=2 blocks
* No. of blocks needed at level n-3, b3 =⎡2/72⎤=1 blocks
* No. Of levels in B tree is 3 (n=2).
* No. of access required here is 3+1=**4**.
* No. of blocks needed for B tree = 84+2+1 = 87
* Additional space for B tree = (84+2+1)KB = **87**KB
* Each record in the file is now effectively using up additional space = (87\*1024)/6000 = **14.84** Bytes/record.

1. Consider the case of **B+ trees:**

* Order of B+ trees in our case is

P\*5+ (p-1)\*4<=1024

Which leads to p=114(maximum value)

Root 1 node 113entries 114pointers

Level1 114nodes 12882 entries 12996pointers

* No. of blocks needed at leaf node (level n) ,b1=⎡6000/113⎤= **54**
* No. Of blocks needed at level n-1 B2=⎡/113⎤=**1**
* No. Of levels in B+ tree is 2 (n=1).
* No. of access required here is 2+1=**3**.
* Total no. of blocks required=54+1=**55**
* Additional space for B+ tree = (54+1)KB = **28**KB
* Additional space for each record is= (55\*1024)/6000 = **9.40** bytes /record

**Comparing storage space needed, no. of blocks accessed by various file organization for Utilizes file:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Total No. of blocks** | **Additional**  **Space/record** | **No. of disk block accesses** |
| Effect of blocking of records with binary search (unspanned) | 24 | 0 bytes | 5 |
| Secondary index  (without multilevel) | 24 + 27(index blocks) = 51 | 10.17 bytes | 7 |
| Secondary index | 24 + 28(index blocks) = 52 | 10.55 bytes | 3 |
| B trees | 24 + 43(B tree blocks) = 67 | 14.84 bytes | 4 |
| B+ trees | 24 + 28 (B+ tree blocks)= 52 | 9.40 bytes | 3 |

So we will be using B+ tree indexing for Cultivates relation since it has the least no. of disk block accesses and least additional space/record.

The total storage space required=26.48KB+28KB (for B+ tree) =**54.48KB**

# NEEDS

|  |  |  |
| --- | --- | --- |
| Cropid | Feid | Total |
| 4B | 4B | 8B |

**General calculations:**

Blocking factor, ⎣1024/8⎦ =1**28**

No. of file blocks required = ⎡000/128⎤ = **8**

**Storage Requirements:**

* By choosing “*unspanned record blocking*” the no. of bytes wasted/block =

1024-128\*8 = **0** Bytes

* This makes each record effectively use 1024/13 = 16 Bytes of storage, hence causing no overhead. [If we use the spanned record blocking it will only create an extra overhead of accessing extra blocks to get a record].
* In addition IBG (Inter Block Gap) uses up to 106 bytes/block.
* Total storage required for Needs file on disk = (Block size + Inter-Block size)\* 8 = (1024+106)\*8= **8.83**KB

**Access Method**

1. Since the number of File blocks required is 8, if we use the binary search on this data file, it would need approx.

[log2 8] = **3** block accesses.

1. If we consider **Secondary indexing(based on any one attribute)**

* To construct secondary index on Cropid (or FeID), the size of index entry is 4 bytes + 5 bytes (for block printer) = **9** bytes.
* Blocking factor is ⎣1024/9⎦ = **113** entries/block.
* Total no. of index entries = **1000**.
* No. of blocks needed for index file = ⎡1000/113⎤ = **9** blocks.
* A binary search on this secondary index needs [log2 9] = **4** block accesses.
* So to search for a record we need, 4+1 (to access the data file block containing record) = **5** block accesses
* Additional size required for secondary indexing(with interblock gap) is **9.93**KB
* Additional space per record is 9.93\*1024/1000 = **10.17**Bytes /record.

1. If we convert secondary index into **multilevel secondary index**,

* Index blocking factor = **113** entries/block.
* Number of 1st level blocks(got from secondary indexing) is 9.
* The no. of 2nd level blocks are ⎡9/113⎤ = 1 block.
* Total blocks required for this is 1+9=**10**.
* Additional size required for secondary indexing with multiplexing(with interblock gap) is **11.04**KB
* No. of block access to access the record is 2+1=**3**.
* So, the additional over head for each record in data file is (11.04\*1024)/1000=**11.30** Bytes/record.

1. Consider the case of **B trees**

* We know, search field (Fid) is 4 bytes long, block pointer = 5 Bytes and Disk block size = 1024 Bytes.

So the order of each node = (p\*5) + (p-1)\*4+ (p-1)\*5<1024

Which leads to p=**73**(consider maximum)

* We can start from root and see how many values and pointers exist in the average at each subsequent level.

Root 1 nodes 72 entries 73 pointers

Level 1 73 nodes 5256entries 5329pointers

Level 2 5329nodes 383688entries 389017pointers

* No. of blocks needed at level ‘n’ , b1 =⎡1000/72⎤= 14
* No. of blocks needed at level n-1, b2 =⎡14/72⎤=1 blocks
* No. Of levels in B tree is 2 (n=1).
* No. of access required here is 2+1=**3**.
* No. of blocks needed for B tree = 14+1 = 15
* Additional space for B tree = (14+1)KB = **15**KB
* Each record in the file is now effectively using up additional space = (15\*1024)/1000 = **15.36** Bytes/record.

1. Consider the case of **B+ trees:**

* Order of B+ trees in our case is

P\*5+ (p-1)\*4<=1024

Which leads to p=114(maximum value)

Root 1 node 113entries 114pointers

Level1 114nodes 12882 entries 12996pointers

* No. of blocks needed at leaf node (level n) ,b1=⎡1000/113⎤= **9**
* No. Of blocks needed at level n-1 B2=⎡/113⎤=**1**
* No. Of levels in B+ tree is 2 (n=1).
* No. of access required here is 2+1=**3**.
* Total no. of blocks required=9+1=**10**
* Additional space for B+ tree = (10+1)KB = **11**KB
* Additional space for each record is= (11\*1024)/1000 = **11.30** bytes /record

**Comparing storage space needed, no. of blocks accessed by various file organization for Utilizes file:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Total No. of blocks** | **Additional**  **Space/record** | **No. of disk block accesses** |
| Effect of blocking of records with binary search (unspanned) | 8 | 0 bytes | 3 |
| Secondary index  (without multilevel) | 8+ 9(index blocks) = 17 | 10.17 bytes | 5 |
| Secondary index | 8 + 10(index blocks) = 18 | 11.30 bytes | 3 |
| B trees | 8 + 15(B tree blocks) = 23 | 15.36 bytes | 3 |
| B+ trees | 8 + 10 (B+ tree blocks)= 18 | 11.30 bytes | 3 |

So we will be using B+ tree indexing for Needs relation since it has the least no. of disk block accesses and least additional space/record.

The total storage space required=8.83KB+18KB (for B+ tree) =**26.83KB**

# INFLUENCED BY

|  |  |  |
| --- | --- | --- |
| Cropid | Locid | Total |
| 4B | 4B | 8B |

**General calculations:**

Blocking factor, ⎣1024/8⎦ =1**28**

No. of file blocks required = ⎡000/128⎤ = **8**

**Storage Requirements:**

* By choosing “*unspanned record blocking*” the no. of bytes wasted/block =

1024-128\*8 = **0** Bytes

* This makes each record effectively use 1024/13 = 16 Bytes of storage, hence causing no overhead. [If we use the spanned record blocking it will only create an extra overhead of accessing extra blocks to get a record].
* In addition IBG (Inter Block Gap) uses up to 106 bytes/block.
* Total storage required for Influenced\_by file on disk = (Block size + Inter-Block size)\* 8 = (1024+106)\*8= **8.83**KB

**Access Method**

1. Since the number of File blocks required is 8, if we use the binary search on this data file, it would need approx.

[log2 8] = **3** block accesses.

1. If we consider **Secondary indexing(based on any one attribute)**

* To construct secondary index on Cropid (or LocID), the size of index entry is 4 bytes + 5 bytes (for block printer) = **9** bytes.
* Blocking factor is ⎣1024/9⎦ = **113** entries/block.
* Total no. of index entries = **1000**.
* No. of blocks needed for index file = ⎡1000/113⎤ = **9** blocks.
* A binary search on this secondary index needs [log2 9] = **4** block accesses.
* So to search for a record we need, 4+1 (to access the data file block containing record) = **5** block accesses
* Additional size required for secondary indexing(with interblock gap) is **9.93**KB
* Additional space per record is 9.93\*1024/1000 = **10.17**Bytes /record.

1. If we convert secondary index into **multilevel secondary index**,

* Index blocking factor = **113** entries/block.
* Number of 1st level blocks(got from secondary indexing) is 9.
* The no. of 2nd level blocks are ⎡9/113⎤ = 1 block.
* Total blocks required for this is 1+9=**10**.
* Additional size required for secondary indexing with multiplexing(with interblock gap) is **11.04**KB
* No. of block access to access the record is 2+1=**3**.
* So, the additional over head for each record in data file is (11.04\*1024)/1000=**11.30** Bytes/record.

1. Consider the case of **B trees**

* We know, search field (Fid) is 4 bytes long, block pointer = 5 Bytes and Disk block size = 1024 Bytes.

So the order of each node = (p\*5) + (p-1)\*4+ (p-1)\*5<1024

Which leads to p=**73**(consider maximum)

* We can start from root and see how many values and pointers exist in the average at each subsequent level.

Root 1 nodes 72 entries 73 pointers

Level 1 73 nodes 5256entries 5329pointers

Level 2 5329nodes 383688entries 389017pointers

* No. of blocks needed at level ‘n’ , b1 =⎡1000/72⎤= 14
* No. of blocks needed at level n-1, b2 =⎡14/72⎤=1 blocks
* No. Of levels in B tree is 2 (n=1).
* No. of access required here is 2+1=**3**.
* No. of blocks needed for B tree = 14+1 = 15
* Additional space for B tree = (14+1)KB = **15**KB
* Each record in the file is now effectively using up additional space = (15\*1024)/1000 = **15.36** Bytes/record.

1. Consider the case of **B+ trees:**

* Order of B+ trees in our case is

P\*5+ (p-1)\*4<=1024

Which leads to p=114(maximum value)

Root 1 node 113entries 114pointers

Level1 114nodes 12882 entries 12996pointers

* No. of blocks needed at leaf node (level n) ,b1=⎡1000/113⎤= **9**
* No. Of blocks needed at level n-1 B2=⎡/113⎤=**1**
* No. Of levels in B+ tree is 2 (n=1).
* No. of access required here is 2+1=**3**.
* Total no. of blocks required=9+1=**10**
* Additional space for B+ tree = (10+1)KB = **11**KB
* Additional space for each record is= (11\*1024)/1000 = **11.30** bytes /record

**Comparing storage space needed, no. of blocks accessed by various file organization for Influenced\_by file:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Total No. of blocks** | **Additional**  **Space/record** | **No. of disk block accesses** |
| Effect of blocking of records with binary search (unspanned) | 8 | 0 bytes | 3 |
| Secondary index  (without multilevel) | 8+ 9(index blocks) = 17 | 10.17 bytes | 5 |
| Secondary index | 8 + 10(index blocks) = 18 | 11.30 bytes | 3 |
| B trees | 8 + 15(B tree blocks) = 23 | 15.36 bytes | 3 |
| B+ trees | 8 + 10 (B+ tree blocks)= 18 | 11.30 bytes | 3 |

So we will be using B+ tree indexing for Influenced\_by relation since it has the least no. of disk block accesses and least additional space/record.

The total storage space required=8.83KB+18KB (for B+ tree) =**26.83KB**

# CROP\_REGISTERED\_FOR

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Fid | Cropid | Inid | Doi | Total |
| 4B | 4B | 4B | 8B | 20B |

**General calculations:**

Blocking factor, ⎣1024/20⎦ =**51**

No. of file blocks required = ⎡8000/25⎤ = **320**

**Storage Requirements:**

* By choosing “*unspanned record blocking*” the no. of bytes wasted/block =

1024-20\*51 = **4** Bytes

* This makes each record effectively use 1024/51 = 20.08 Bytes of storage. So the overhead is only 0.08 Bytes/record. [If we use the spanned record blocking we can save this but it leads to complexity in accessing blocks, like sometimes we need to access 2 blocks to get a record].
* In addition IBG (Inter Block Gap) uses up to 106 bytes/block.
* Total storage required for Crop\_registered\_for file on disk = (Block size + Inter-Block size)\* 320 = (1024+106)\*320= **353.12**KB

**Access Method**

1. Since the number of File blocks required is 320, if we use the binary search on this data file, it would need approx.

[log2 320] = **9** block accesses.

1. If we consider **Secondary indexing(based on any one attribute)**

* To construct secondary index on Fid (or Cropid / Inid), the size of index entry is 4 bytes + 5 bytes (for block printer) = **9** bytes.
* Blocking factor is ⎣1024/9⎦ = **113** entries/block.
* Total no. of index entries = **8000**.
* No. of blocks needed for index file = ⎡8000/113⎤ = **71** blocks.
* A binary search on this secondary index needs [log2 71] = **7** block accesses.
* So to search for a record we need, 7+1 (to access the data file block containing record) = **8** block accesses
* Additional size required for secondary indexing(with interblock gap) is **78.35**KB
* Additional space per record is 78.35\*1024/8000 = **10.03**Bytes /record

1. If we convert secondary index into **multilevel secondary index**,

* We know index blocking factor = **113** entries/block.
* Therefore the no. of 1st level blocks is 71.
* The no. of 2nd level blocks are ⎡71/113⎤ = 1 block.
* Total blocks required for this is 1+71=**72**.
* So, disk space required is 79.45KB.
* No. of block access to access the record is 2+1=**3**.
* So, the additional over head for each record in data file is (79.45\*1024)/8000 = **10.17** Bytes/record.

1. Consider the case of **B trees**

* We know, search field (Fid) is 4 bytes long, block pointer = 5 Bytes and Disk block size = 1024 Bytes.

So the order of each node = (p\*5) + (p-1)\*4+ (p-1)\*5<1024

Which leads to p=**73**(consider maximum)

* We can start from root and see how many values and pointers exist in the average at each subsequent level.

Root 1 nodes 72 entries 73 pointers

Level 1 73 nodes 5256entries 5329pointers

Level 2 5329nodes 383688entries 389017pointers

* No. of blocks needed at level ‘n’ , b1 =⎡8000/72⎤= 112
* No. of blocks needed at level n-1, b2 =⎡112/72⎤=2 blocks
* No. of blocks needed at level n-2, b3 =⎡2/72⎤=1 blocks
* No. Of levels in B tree is 3 (n=2).
* No. of access required here is 3+1=**4**.
* No. of blocks needed for B tree = 112+2+1 = 115
* Additional space for B tree = (115+2+1)KB = **115**KB
* Each record in the file is now effectively using up additional space = (114\*1024)/8000 = **14.72** Bytes/record.

1. Consider the case of **B+ trees:**

* Order of B+ trees in our case is

P\*5+ (p-1)\*4<=1024

Which leads to p=114(maximum value)

Root 1 node 113entries 114pointers

Level1 114nodes 12882 entries 12996pointers

* No.of blocks needed at leaf node (level n) ,b1=⎡8000/113⎤= **71**
* No. Of blocks needed at level n-1 B2=⎡/113⎤=**1**
* No. Of levels in B+ tree is 2 (n=1).
* No. of access required here is 2+1=**3**.
* Total no. of blocks required=71+1=**72**
* Additional space for B+ tree = (71+1)KB = **72**KB
* Additional space for each record is= (72\*1024)/300 = **9.22** bytes /rec

**Comparing storage space needed, no. of blocks accessed by various file organization for Crop\_registered\_for file:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Total No. of blocks** | **Additional**  **Space/record** | **No. of disk block accesses** |
| Effect of blocking of records with binary search (unspanned) | 320 | 0.08 bytes | 9 |
| Secondary index  (without multilevel) | 320 + 71(index blocks) = 391 | 10.03 bytes | 8 |
| Secondary index  (with multilevel) | 320 + 72(index blocks) = 392 | 10.17 bytes | 3 |
| B trees | 320 + 115(B tree blocks) = 435 | 14.72 bytes | 4 |
| B+ trees | 320 + 72(index blocks) = 392 | 9.22 bytes | 3 |

So we will be using B+ trees for Crop\_required\_for relation since it has the least no. of disk block along with lowest additional space requirement.

The total storage space required=353.12KB+4KB (for B+ tree) =**357.12KB**

# EQUIP\_REGISTERED\_FOR

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Fid | Name | Type | Inid | Doi | Total |
| 4B | 20B | 10B | 4B | 8B | 46B |

**General calculations:**

Blocking factor, ⎣1024/46⎦ =**22**

No. of file blocks required = ⎡5000/22⎤ = **228**

**Storage Requirements:**

* By choosing “*unspanned record blocking*” the no. of bytes wasted/block =

1024-22\*46 = **12** Bytes

* This makes each record effectively use 1024/22 = 46.55 Bytes of storage. So the overhead is only 0.55 Bytes/record. [If we use the spanned record blocking we can save this but it leads to complexity in accessing blocks, like sometimes we need to access 2 blocks to get a record].
* In addition IBG (Inter Block Gap) uses up to 106 bytes/block.
* Total storage required for equip\_registered\_for file on disk = (Block size + Inter-Block size)\* 228 = (1024+106)\*228= **351.6**KB

**Access Method**

1. Since the number of File blocks required is 320, if we use the binary search on this data file, it would need approx.

[log2 228] = **8** block accesses.

1. If we consider **Secondary indexing(based on any one attribute)**

* To construct secondary index on Fid (Since it has least size), the size of index entry is 4 bytes + 5 bytes (for block printer) = **9** bytes.
* Blocking factor is ⎣1024/9⎦ = **113** entries/block.
* Total no. of index entries = **5000**.
* No. of blocks needed for index file = ⎡5000/113⎤ = **45** blocks.
* A binary search on this secondary index needs [log2 45] = **6** block accesses.
* So to search for a record we need, 6+1 (to access the data file block containing record) = **7** block accesses
* Additional size required for secondary indexing(with interblock gap) is **49.66**KB
* Additional space per record is 49.66\*1024/5000 = **10.17**Bytes /record

1. If we convert secondary index into **multilevel secondary index**,

* We know index blocking factor = **113** entries/block.
* Therefore the no. of 1st level blocks is 45.
* The no. of 2nd level blocks are ⎡45/113⎤ = 1 block.
* Total blocks required for this is 1+45=**46**.
* So, disk space required is 50.76KB.
* No. of block access to access the record is 2+1=**3**.
* So, the additional over head for each record in data file is (50.76\*1024)/5000 = **10.39** Bytes/record.

As per profit/loss calculation and query processing is concerned there is no need to go for cluster indexing because grouping is done using primary key in any case.

1. Consider the case of **B trees**

* We know, search field (Fid) is 4 bytes long, block pointer = 5 Bytes and Disk block size = 1024 Bytes.

So the order of each node = (p\*5) + (p-1)\*4+ (p-1)\*5<1024

Which leads to p=**73**(consider maximum)

* We can start from root and see how many values and pointers exist in the average at each subsequent level.

Root 1 nodes 72 entries 73 pointers

Level 1 73 nodes 5256entries 5329pointers

Level 2 5329nodes 383688entries 389017pointers

* No. of blocks needed at level ‘n’ , b1 =⎡5000/72⎤= 70
* No. of blocks needed at level n-1, b2 =⎡70/72⎤=1 blocks
* No. Of levels in B tree is 2 (n=1).
* No. of access required here is 2+1=**3**.
* No. of blocks needed for B tree = 70+1 = 71
* Additional space for B tree = (70+1)KB = **71**KB
* Each record in the file is now effectively using up additional space = (71\*1024)/5000 = **14.54** Bytes/record.

1. Consider the case of **B+ trees:**

* Order of B+ trees in our case is

P\*5+ (p-1)\*4<=1024

Which leads to p=114(maximum value)

Root 1 node 113entries 114pointers

Level1 114nodes 12882 entries 12996pointers

* No.of blocks needed at leaf node (level n) ,b1=⎡5000/113⎤= **45**
* No. Of blocks needed at level n-1 B2=⎡/113⎤=**1**
* No. Of levels in B+ tree is 2 (n=1).
* No. of access required here is 2+1=**3**.
* Total no. of blocks required=45+1=**46**
* Additional space for B+ tree = (45+1)KB = **46**KB
* Additional space for each record is= (46\*1024)/5000 = **9.42** bytes /rec

**Comparing storage space needed, no. of blocks accessed by various file organization for Crop\_registered\_for file:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Total No. of blocks** | **Additional**  **Space/record** | **No. of disk block accesses** |
| Effect of blocking of records with binary search (unspanned) | 228 | 0.55 bytes | 8 |
| Secondary index  (without multilevel) | 228 + 45(index blocks) = 273 | 10.17 bytes | 7 |
| Secondary index  (with multilevel) | 228 + 46(index blocks) = 274 | 10.39 bytes | 3 |
| B trees | 228 + 71(B tree blocks) = 299 | 14.54 bytes | 3 |
| B+ trees | 228 + 46(index blocks) = 273 | 9.42 bytes | 3 |

So we will be using B+ trees for equip\_required\_for relation since it has the least no. of disk block along with lowest additional space requirement.

The total storage space required=351.6KB+46KB (for B+ tree) =**397.6KB**

D] Timings

Time required to access record in each table

Access time=(No. of block access) \* (seek time(s) + latency time(l) + block transfer time(btt))

Optimizations:

For indexing the index blocks can be stored on the same cylinder on the disk reducing the seek time.

So optimised time can be calculated as

Access time=(1\* seek time(s)) +((No. of block access) \* ( latency time(l) + block transfer time(btt)))

For multilevel indexing, the top level (first) index blocks can be stored in the main memory itself, reducing the number of block access required.

For B/B+ trees the root node can be stored in the main memory itself.

So optimised time can be calculated as

Access time =(No. of block access-1) \* (seek time(s) + latency time(l) + block transfer time(btt))

The optimised access times for the different file organisations are calculated as stated above.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| RELATIONS | FILE ORGANISATION | BLOCK ACCESS | ACCESS TIME(ms) | OPTIMISED ACCESS TIME(ms) |
| FARMER | B+ TREE | 4 | 120 | 60 |
| FARMER\_INS | B+ TREE | 3 | 90 | 40 |
| FARMER\_LOAN | B+ TREE | 3 | 90 | 40 |
| CROP | PRIMARY | 2 | 60 | 40 |
| FERTILIZER | SECONDARY | 3 | 90 | 50 |
| LOCATION | B+ TREE | 3 | 90 | 40 |
| LOCATION\_SOIL | B+ TREE | 2 | 60 | 20 |
| LOAN | SECONDARY | 2 | 60 | 40 |
| INSURANCE | SECONDARY | 2 | 60 | 40 |
| FARM\_EQUIPMENT | SECONDARY | 3 | 90 | 50 |
| UTILIZES | B+ TREE | 3 | 90 | 40 |
| CULTIVATES | B+ TREE | 3 | 90 | 40 |
| NEEDS | B+ TREE | 3 | 90 | 40 |
| INFLUENCED\_BY | B+ TREE | 3 | 90 | 40 |
| CROP\_REGISTERED\_FOR | B+ TREE | 3 | 90 | 40 |
| EQUIP\_REGISTERED\_FOR | B+ TREE | 3 | 90 | 40 |

Buffering

For B/B+ trees the root node is stored in the main memory itself. Hence the buffer space should be enough to store all the root nodes of such relation.

As listed above , 11 tables have B+ tree file organisation ,hence total buffer space

=11\*1024=**11KB**

Time required for each queries

1. Farmer may want to view his information and update it

* Files needed : Farmer
* Type of File Organization used : B+tree (Farmer)
* Number of block access required : 3
* Access time = (No. of block access) \* (seek time(s) + latency time(l) + block transfer time(btt)) = 4\*(20+9.5+0.5)ms=120ms
* Optimised access time =60ms

2. Search a financial scheme or loan available to farmers by name / income range

* Files needed : Loan
* Type of File Organization used : Secondary Indexing
* Number of block access required : 2
* Access time = (No. of block access) \* (seek time(s) + latency time(l) + block transfer time(btt)) = 2\*(20+9.5+0.5)ms=60ms
* Optimised access time =40ms

3. Get information of a particular crop with suitable fertilizer and soil type

* Files needed : Crop, Needs , Fertilizer
* Type of File Organization used : primary indexing (Crop file)**,** B+ Tree(Needs file), Secondary Indexing(Fertilizer file)
* Number of block access required : 2 + 3 + 3 = 8
* Access time = (No. of block access) \* (seek time(s) + latency time(l) + block transfer time(btt)) =8\*(20+9.5+0.5)ms = 240ms
* Optimised access time =130ms

4. Information on which crops are being cultivated based on income range of the farmer

* Files needed : Crop, Farmer, Cultivates
* Type of File Organization used : primary indexing(Farmer file), Primary Indexing(Crop file), B+ Tree(Cultivates file)
* Number of block access required : 4 + 2 + 3
* Access time = (No. of block access) \* (seek time(s) + latency time(l) + block transfer time(btt)) = 9\*(20+9.5+0.5)ms = 270ms
* Optimised access time =140ms

5. The data entry user can search for crops / fertilizers/ loan schemes to update / delete the data

* Files needed : Crop
* Type of File Organization used : Primary Indexing(Crop file)
* Number of block access required : 2
* Access time = (No. of block access) \* (seek time(s) + latency time(l) + block transfer time(btt)) = 2\*(20+9.5+0.5)ms = 60ms
* Optimised access time =40ms

6. Information about which crop is suitable to be grown based on farmer’s location, soil type, weather condition

* Files needed : Farmer, Influenced\_By, Location
* Type of File Organization used : Primary Indexing(Farmer), B+ Tree(Location), B+ Tree(Influenced\_By)
* Number of block access required : 4 + 3 + 3 = 10
* Access time = (No. of block access) \* (seek time(s) + latency time(l) + block transfer time(btt)) = 10\*(20+9.5+0.5)ms = 300ms
* Optimised access time=140ms

7. Information on various insurance schemes which the farmers of a certain income range have availed for their farm equipment and crops

* Files needed : Farmer,Crop\_Registered,Equip\_Registered,Insurance
* Type of File Organization used : Primary Indexing(Farmer), B+ Tree(Crop\_Registered), B+ Tree(Equip\_Registered), Secondary Indexing(Insurance)
* Number of block access required : 4 + 3 + 3 + 2
* Access time = (No. of block access) \* (seek time(s) + latency time(l) + block transfer time(btt)) = 12\*(20+9.5+0.5)ms = 360ms
* Optimised access time=180ms

E] Work Space Requirements

Work space requirement depends on the operations out of which join is the costliest one.

Consider the join with maximum number of tables(Query no.7).

Tables required are : Farmer,Crop\_Registered,Equip\_Registered,Insurance

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tables | Farmer | Crop\_registered\_for | Equip\_registered\_for | Insurance |
| Blocks | 743 | 320 | 228 | 4 |

The final result can have a tuple of maximum size (76+38+62) = 176B

Let the buffers be available for all tables as

Farmer : 7

Crop\_registered\_for:3

Equip\_registered\_for:3

Insurance:2

Result:15

Total buffers required :30

Hence work space required: **30KB**

Considering all the attributes listed in the queries,

All attributes of Farmer file = 76B

All attributes of Crop file = 68B

All attributes of Loan file = 38B

All attributes of Insurance file = 62B

Attributes Fid, Inid of Crop\_Registered\_For file = 8B

Attributes Fid, Inid of Equip\_Registered\_For file = 8B

All attributes of Influenced\_By file = 8B

Attributes Cropid, Feid from Needs file = 8B Attribute Name from Fertilizer file = 10B

Attributes cropid from Cultivates file = 4B

Attributes district, state from Location file = 20B

Total = 310B

F] System Specifications

Total disc storage space required is

Table Space Required (KB)

1. FARMER 910
2. FARMER\_INS 158.73
3. FARMER\_LOAN 158.73
4. CROP 38.51
5. FERTILIZER 12.13
6. LOCATION 17.24
7. LOCATION\_SOIL 38.79
8. LOAN 3.31
9. INSURANCE 5.41
10. FARM\_EQUIPMENT 7.65
11. UTILIZES 54.48
12. CULTIVATES 54.48
13. NEEDS 26.83
14. INFLUENCED\_BY 26.83
15. CROP\_REGISTERED\_FOR 357.12
16. EQUIP\_REGISTERED\_FOR 397.6

TOTAL 2267.87

Total memory space required=Work space + buffer

=30KB+11KB

=41KB