## Chapter 3.7 Bitmap Index

***Principle:***

Assuming that the file records have a permanent identification: 1, 2, 3..., n. There exists some data structure, for random i, enables us to find the ith record easily.

***Definition:***

The bitmap for field F is a byte vector collection with length n. Each byte corresponds to a possible value that may exists or not.

1. If the ith record of field F equals to v, then in byte vector collection, it assigns to 1 when the field F of records equals to v.
2. If the ith record of field F doesn’t equal to v, then in the byte vector collection, it assigns to 0 when the field F of records doesn’t equal to v.

***Example:***

Assuming that the record in file consist of two fields F and G, and the field F is integer and the field G is string. Now, there exist six records in the file, the values are (30, foo), (30, bar), (40, baz), (50, foo), (40, bar), (30, baz).

* *Field F*

|  |  |
| --- | --- |
| Possible Values of Field F | Each Byte indicates Whether ith Record equals to Current Possible Value of Field F |
| 30 | 110001 |
| 40 | 001010 |
| 50 | 000100 |

* *Field G*

|  |  |
| --- | --- |
| Possible Values of Field G | Each Byte indicates Whether ith Record equals to Current Possible Value of Field G |
| foo | 100100 |
| bar | 010010 |
| baz | 001001 |

### Chapter 3.7.1 Motivation of Bitmap Index

It seems that the bitmap index needs a lot of spaces, especially when the field needs plenty of different values, and the total number of the byte is the value of record multiple by the number of values.

*(For example, if the field is the key and the number of records equals to n, then the byte vector needs n\*n byte.)*

***Scenario1: Partial Matching Searching***

Bitmap index enables us to answer the partial matching searching question with high efficiency.

***Example1:***

SELECT title FROM Movie

WHERE studioName = ‘Disney’ AND year = 2005;

Assuming that there have Bitmap index on the fields studioName and year, then we calculate the intersection vector of year = 2005 and studioName = ‘Disney’. After finishing the calculation, the ith location equals to 1, and only when the corresponding movie in ith Movie tuple is made by Disney in the year 2005.

***Scenerio2: Ranging Searching***

Bitmap index can also help us answer the ranging search. In the example below, it not only state the usage in ranging search but also explain how to use the vector to search the answer, also it only used to search the needed record but not search other records.

***Example2:***

Assuming that twelve points numbered from 1 to 12:

|  |  |  |  |
| --- | --- | --- | --- |
| 1: (25, 60) | 2: (45, 60) | 3: (50, 75) | 4: (50, 100) |
| 5: (50, 120) | 6: (70, 110) | 7: (85, 140) | 8: (30, 260) |
| 9: (25, 400) | 10: (45, 350) | 11: (50, 275) | 12: (60, 260) |

*Field Age:*

|  |  |
| --- | --- |
| Possible Value Exists on Field Age | Each Byte indicates Whether ith Record equals to Current Possible Value of Field Age |
| 25 | 100000001000 |
| 30 | 000000010000 |
| 45 | 010000000100 |
| 50 | 001110000010 |
| 60 | 000000000001 |
| 70 | 000001000000 |
| 85 | 000000100000 |

*Field Salary:*

|  |  |
| --- | --- |
| Possible Value Exists on Field Salary | Each Byte indicates Whether ith Record equals to Current Possible Value of Field Salary |
| 60 | 110000000000 |
| 75 | 001000000000 |
| 100 | 000100000000 |
| 110 | 000001000000 |
| 120 | 000010000000 |
| 140 | 000000100000 |
| 260 | 000000010001 |
| 275 | 000000000010 |
| 350 | 000000000100 |
| 400 | 000000001000 |

Assuming that we need to find all buyers who are in age range (45, 55) and salary range (100K, 200K).

* Find age equals to 45, then bitmap value equals to 010000000100.
* Find age equals to 50, then bitmap value equals to 001110000010.
* 010000000100 || 001110000010 = 011110000110.
* Find salary equals to 100K, then bitmap value equals to 000100000000.
* Find salary equals to 110K, then bitmap value equals to 000001000000
* Find salary equals to 120K, then bitmap value equals to 000010000000.
* Find salary equals to 140K, then bitmap value equals to 000000100000.
* 000100000000 || 000001000000 || 000010000000 || 000000100000 = 000111100000
* Finally, using 011110000110 & 000111100000 = 0001100000 to represent the final bitmap value in age range (45, 55) and salary range (100K, 200K).
* Convert bitmap value and find only the forth and fifth points *4: (50, 100)* and *5: (50, 120)* are satisfied the condition.

### Chapter 3.7.2 Compressed Bitmap

***Principle:***

Assuming that there has a file that contains n records. In those records, the field F has m different values. All binary byte numbers of byte vectors of such index is m\*n. So if the size of block is 4096 bytes, then 32768 bytes can be stored in one block, so storing m\*n bytes in such block equals to m\*n/32768. The number of block that required is much less than store the block itself, but as m turns bigger, then the bitmap will require more space.

***Encode Solution:***

Segment length encode is one common solution. Segment consists of a sequence which has *i number 0 and one number 1*. Using some kind of binary encode the integer i to represent segment. After that concatenate each segment, then the byte sequence can be used to present the whole byte vector.

* Make sure how many bytes - j can be used to present i.
* Use j – 1 one and 1 zero to present the first part of byte sequence.

( j – 1 + 1 = j )

* Add the binary number of i after the first part of byte sequence.

*Attention! The binary number can not be used as byte sequence directly.*

*Assume that the byte vector 000101 consists of two segments and one of which is 0001 and another one is 01. In the first segment, i equals to 3 and in the second one i equals to 1. According to encode rule, 3 equals to 11 by using binary number while 1 equals to 1 by using binary number. So the binary number of byte vector 000101 equals to 111. In the same way, byte vector 010001 can be encoded as 111; Also 010101 is also encoded as 111. So 111 can not be the only vector that present byte vector.*

***Example:***

If i = 13, j = 4. So we need 4 bytes to present i.

* The first step: log2(13) = 4, so we need 4 bytes to present i.
* The second step: j = 4 – 1, so the first part of byte sequence equals to 1110.
* The third step: 13 -> 1101, so the second part of byte sequence equals to 1101.
* The forth step: 1110 + 1101 -> at last, the whole part of byte sequence equals to 11101101.

***Attention:***

* i = 1, then the binary number equals to 01 while i = 0, then the binary number equals to 00.
* Under these two situations, j = 1. So i = 1, then it encoded as 01; In the same way, i = 0, then it encoded as 00.

***Decode Principle:***

If we already have the integer sequence, then we can recover the length sequence of the whole segment.

* Assume that we are at the situation that starts from the coding sequence. Scan the first zero and make sure the value of j and the value j equals to the byte numbers that start with the first 0.
* After getting the value j, then j byte binary number equals to i. Once finished scanning the binary bytes of i, then we get the next location of the integer encoding.

Example:

Decode the sequence 11101101001011.

Example:

### Chapter 3.7.3 Operations On Fractional Length Encoding Bit Vector

### Chapter 3.7.4 Management of Bitmap Index