## 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 the 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 the 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 | 1, 00000001, 000 |
| 30 | 00000001, 0000 |
| 45 | 01, 00000001, 00 |
| 50 | 001, 1, 1, 000001, 0 |
| 60 | 000000000001 |
| 70 | 000001, 000000 |
| 85 | 0000001, 00000 |

*Field Salary:*

|  |  |
| --- | --- |
| Possible Value Exists on Field Salary | Each Byte indicates Whether ith Record equals to Current Possible Value of Field Salary |
| 60 | 1, 10000000000 |
| 75 | 001, 000000000 |
| 100 | 0001, 00000000 |
| 110 | 000001, 000000 |
| 120 | 00001, 0000000 |
| 140 | 0000001, 00000 |
| 260 | 00000001, 0001 |
| 275 | 00000000001, 0 |
| 350 | 0000000001, 00 |
| 400 | 000000001, 000 |

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 01, 00000001, 00.
* Find age equals to 50, then bitmap value equals to 001, 1, 1, 000001, 0.
* 010000000100 || 001110000010 = 011110000110.
* Find salary equals to 100K, then bitmap value equals to 0001, 00000000.
* Find salary equals to 110K, then bitmap value equals to 000001, 000000.
* Find salary equals to 120K, then bitmap value equals to 00001, 0000000.
* Find salary equals to 140K, then bitmap value equals to 0000001, 00000.
* 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, then we need 4 bytes to present i.
* The second step: j = 4 – 1, then the first part of byte sequence equals to 1110.
* The third step: 13 -> 1101, then the second part of byte sequence equals to 1101.
* The forth step: 1110 + 1101 -> at last, and 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. *(1 corresponds to 01 and 0 corresponds to 00.)*
* Under these two situations, j = 1. *(This means that there has zero 1 and one 0.)*

***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 1110, 1101, 00, 10, 11:

1. From sequence 11101101 00 1011, we find that there have 3 bytes 1 and 1 byte 0, and the next four bytes are 1101 which represents 13, so i equals to 13. This is the first binary sequence.
2. Then we find that the binary sequence starts with 0, so the next 0 is the original number. This is the second binary sequence.
3. At last, we find that the first 0 is located at the second place. So j equals to 2 and 11 represents the binary number of i. At last, 11 stands for 3.

*The whole segment length sequence equals to 13, 0 and 3.*

From the whole segment length sequence, we can construct the *actual byte vector* which equals to *0000000000001, 1, 001*.

(Technically, each byte vector should be end with 1, and each sequence end with 0 can not be recovered. But since we know the number of record of the file, then the attached 0 sequence can be added.)

***Example:***

Convert the byte vector to the segment length sequence. When the age equals to 25, 30 and 45, then the byte vectors are 1, 00000001, 000; 00000001, 0000 and 01, 0000001, 00.

1. 1, 00000001, 000:

|  |  |  |  |
| --- | --- | --- | --- |
|  | The First Part | The Second Part | The Third Part |
| Byte Vector | 1 | 00000001 | 000 |
| i -> Binary Value | 0 | 7 -> 111 | ignored |
| j -> Binary Sequence | 0 | 3 -> 110 | ignored |
| Partial Sequence | 00 | 110, 111 | ignored |
| Final Total Sequence | 00, 110, 111 | | |

1. 00000001, 0000:

|  |  |  |
| --- | --- | --- |
|  | The First Part | The Second Part |
| Byte Vector | 00000001 | 0000 |
| i -> Binary Vector | 7 -> 111 | ignored |
| j -> Binary Sequence | 3 -> 110 | ignored |
| Partial Sequence | 100, 111 | ignored |
| Final Total Sequence | 100, 111 | |

1. 01, 00000001, 00:

|  |  |  |  |
| --- | --- | --- | --- |
|  | The First Part | The Second Part | The Third Part |
| Byte Vector | 01 | 00000001 | 00 |
| i -> Binary Vector | 1 | 7 -> 111 | ignored |
| j -> Binary Sequence | 0 | 3 -> 110 | ignored |
| Partial Sequence | 01 | 110, 111 | ignored |
| Final Total Sequence | 01, 110, 111 | | |

***Conclusion:***

Assume that m = n, which means that the property of the whole bitmap just exists for one time. Since there has n byte vector in the index, then the total length of the index at most equals to 2nlog2n, without compression, it requires n\*n bytes.

### Chapter 3.7.3 Operations On Segment Length Encoding Byte Vector

***Principle:***

When we need to operate and/or on the byte vector, then we just need to decode them and get the original byte vector. But we do not need to decode all the byte vector.

The compression method makes us to *decode one segment once a time*, therefore we can make sure *the location of next one in the byte vector*.

* *OR operation*, if exists 1, then in the corresponding location output 1.
* *AND operation*, then only if two operation objects has the next 1 in the same location, then generate 1.

***Example:***

The encoding byte vector of age = 25 equals to 00110111 and the encoding byte vector of age = 30 equals to 110111.

1. 00, 110, 111:

|  |  |  |
| --- | --- | --- |
|  | The First Part | The Second Part |
| Segment Sequence | 00 | 110, 111 |
| j | 0 | 3 |
| i | 0 | 7 |
| Original Sequence | 1, 00000001 | |

1. 110, 111:

|  |  |
| --- | --- |
|  | The First Part |
| Segment Sequence | 110, 111 |
| j | 3 |
| i | 7 |
| Original Sequence | 00000001 |

Operate AND on two sequences, 100000001 AND 00000001, get final sequence 100000010.

***Attention:***

Since all uncompressed sequences are all with the length of 12, so technically, the final sequence equals to 100000010.

### Chapter 3.7.4 Management of Bitmap Index

***Instructions:***

We already describe the operations on the bitmap index, so there still three more questions are needed to be discussed:

1. When we want to search a byte vector by giving a specific value, or multi - byte vectors in the given range, then how can we find them efficiently?
2. How could we find the record efficiently when we already find the collection for the question?
3. When there have some changes on the data insertion or deletion of the data file, how could we adjust these byte index of the given field?

***Principle:***

*Find Byte Vector*

* See the byte vector as records, and their key is field value. Then any auxiliary index can help us find these byte vectors based on the key values efficiently.
* We also need to store byte vector somewhere. It’s better to see them as the variable length records, since as the number of records increased, then their length will also increase.

*Find Records*

Once we make sure that the record k in the file, then how could we find them. Just see the kth record as the index key value k. Then we can create auxiliary index in the data file, and the record number is their index key.

*Modify Data File*

In the bitmap index, the question related with data file modification has two sides:

* Once assigned, then the number of record should be kept.
* The modification on the data file will definitely change the bitmap index.

The consequence of the first one point is that when we delete the record i, then it’s easiest to hide the record number. It should be replaced by the “deleted mark”. The bitmap index should also be changed, since 1 on the location i should be changed as 0.

*Insertion of new records:*

Keep the next available record number, and assign it to the new record. For every bitmap index, we need to make sure the value of new record and add 1 after the value of byte vector. If we add a new 0 after the byte vector, the compressed value does not need to do any change.

*Special situation:*

If there exists one field that has never exist before, then we need to give this value a new byte vector, and this byte vector and it’s related value needs to be inserted into the auxiliary index structure.

*(This structure is used to find the byte vector based on the given value.)*

*Modification of new records:*

When we change the field record from v to w, then we must change the vector of v and change i from 1 to 0.

* If there exists a byte vector with value w, then change the ith location from 0 to 1;
* If there does not exist the byte vector of value w, then we need to create the byte vector.