## **BITMAP INDEXES**

*CS 564- Spring 2025* 

# WHAT IS THIS LECTURE ABOUT?

- Bitmap Indexes
- Bitslice Indexes

### **MOTIVATION**

#### Consider the following table:

#### How can we speed up the following query?

```
SELECT * FROM Tweets
WHERE zip BETWEEN 53000 AND 54999 ;
```

### B+ tree on attribute zip

#### **MOTIVATION**

#### Consider the following table:

How many bytes does a B+ tree use for each record?

- at least key + rid, so: key-size + rid-size
   Can we do better than that (in terms of storage overhead)?
- yes! especially when the attribute domain is small

# **BITMAP INDEX**

### THE BITMAP INDEX

- Consider building an index to answer equality queries on the retweet attribute
- Issues with building a B+ tree:
  - three distinct values: yes, no, NULL
  - many duplicates for each distinct value
  - a weird B+ tree with three long rid lists
- bitmap index: build three bitmap arrays (stored on disk), one for each value
  - the i<sup>th</sup> bit in each bitmap corresponds to the i<sup>th</sup> tuple
     (we need to map the i<sup>th</sup> position to a rid!)

### **BITMAP: EXAMPLE**

#### table (stored in heapfile)

uniqueMsgID	 zip	retweet
1	 11324	yes
2	 53705	yes
3	 53706	no
4	 53705	NULL
5	 90210	no
1,000,000,000	 53705	yes

#### bitmap index (on retweet)

yes	no	
1	0	
1	0	
0	1	
0	0	
0	1	
1	0	

null
0
0
0
1
0
0

SELECT \* FROM Tweets WHERE retweet = "no";

- scan the "no" bitmap file
- for each bit set to 1, compute the tuple rid
- fetch the tuple

## A CRITICAL ISSUE

- We need an efficient way to compute a bit position:
  - layout the bitmap in page-id order
- We need an efficient way to map a bit position to a rid:
  - fix the # records per page in the heapfile
  - lay the pages out so that page-ids are sequential and increasing
  - then construct rid (page-id, slot#)
    - page-id = bit-position / #records-per-page
    - slot# = bit-position % #records-per-page

With variable length records, we have to set the limit based on the size of the largest record, which may result in under-filled pages!

# BITMAP: OTHER QUERIES

#### table (stored in heapfile)

uniqueMsgID	 zip	retweet
1	 11324	yes
2	 53705	yes
3	 53706	no
4	 53705	NULL
5	 90210	no
1,000,000,000	 53705	yes

#### bitmap index (on retweet)

yes	no
1	0
1	0
0	1
0	0
0	1
1	0

11
null
0
0
0
1
0
0

```
SELECT COUNT(*) FROM Tweets WHERE retweet = "no";
SELECT * FROM Tweets WHERE retweet IS NOT NULL;
```

### STORING A BITMAP INDEX

- One bitmap for each value, and one for NULL
- to store each bitmap, use one file for each
- Bitmaps can be compressed!

index size = #tuples \* (domain size + 1) bits

When is a bitmap more space efficient than a B+ tree? #distinct values < data entry size in the B+ tree

# **BITSLICE INDEX**

### **MOTIVATION**

#### Reconsider the following table:

Building a bitmap index on zip is not a good idea!

## **BITSLICE INDEX**

table (stored in heapfile)

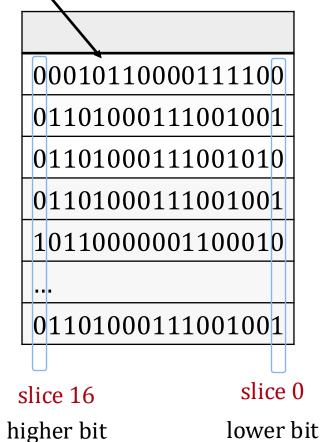
convert to binary

bitslice index

uniqueMsgID	 zip	retweet
1	 11324	yes
2	 53705	yes
3	 53706	no
4	 53705	NULL
5	 90210	no
	 	•••
1,000,000,000	 53705	yes

1 slice per bit

+ (possibly) one more slice for NULL



# **BITSLICE INDEX: QUERIES**

•••	zip
	11324
	53705
	53706
	53705
	90210
	53705

00010110000111100
01101000111001001
01101000111001010
01101000111001001
10110000001100010
01101000111001001
00010111011100000

SELECT \* FROM Tweets
WHERE zip <= 12000;</pre>

**= 12000** in binary

slice 16

slice 0

walk through each slice constructing a result bitmap

- If we look for 0 and have 1, put 0 in the result
- If we look for 1 and have 0, put 1 in the result
- Else we need to consider the next bitslice

## **OTHER QUERIES**

- We can also do aggregates with bitslice indices:
  - e.g. SUM(attr): add bitslice by bitslice
    - count the number of 1s in slice 16 and multiply the count by  $2^{16}$
    - count the number of 1s in slice 15 and multiply the count by  $2^{15}$

•

• We can store each slice using methods like what we have for a bitmap (we can compress again!)

## BITMAP VS BITSLICE INDEX

- Bitmaps are better for low cardinality domains
- Bitslices are better for high cardinality domains

It is generally easier to "do the math" with bitmap indices