Motivation

Consider the following table:

```sql
CREATE TABLE Tweets (  
    uniqueMsgID INTEGER,       -- unique message id  
    tstamp TIMESTAMP,          -- when was the tweet posted  
    uid INTEGER,               -- unique id of the user  
    msg VARCHAR (140),         -- the actual message  
    zip INTEGER,               -- zipcode when posted  
    retweet BOOLEAN            -- retweeted?  
);
```

In the past, we have used a B+-tree for the uid and the zip values.

In a B+-tree, how many bytes do we use for each record?

Can we do better, i.e. an index with lower storage overhead? Especially for attributed with small domain cardinalities?

*Bit-based indices: Two flavors*

a) Bitmap indices and

b) Bitslice indices
Bitmap Indices

• Consider building an index to answer equality queries on the retweet attribute

• Issues with building a B-tree:
  – Three distinct values: True, False, NULL
  – Lots of duplicates for each distinct value
  – Sort of an odd B-tree with three long rid lists

• Bitmap Index: Build three bitmap arrays (stored on disk), one for each value.
  – The i\textsuperscript{th} bit in each bitmap correspond to the i\textsuperscript{th} tuple
    (need to map i\textsuperscript{th} position to a rid)
SELECT * FROM Tweets WHERE retweet = 'N'

1. Scan the R-No Bitmap file
2. For each bit set to 1, compute the tuple #
3. Fetch the tuple # (s)
Critical Issue

• Need an efficient way to compute a bit position
  – Layout the bitmap in page id order.
• Need an efficient way to map a bit position to a record id.

How?

1. If you fix the # records per page in the heapfile
2. And lay the pages out so that page #s are sequential and increasing
3. Then can construct \( \text{rid (page-id, slot#)} \)
   • \( \text{page-id} = \text{Bit-position} / \#\text{records-per-page} \)
   • \( \text{slot#} = \text{Bit-position} \% \#\text{records-per-page} \)

Implications of #1?
## Other Queries

### Table (stored in a heapfile)

<table>
<thead>
<tr>
<th>uniqueMsgID</th>
<th>...</th>
<th>zip</th>
<th>retweet</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>11324</td>
<td>Y</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>53705</td>
<td>Y</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>53706</td>
<td>N</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>53705</td>
<td>NULL</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>90210</td>
<td>N</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1,0000000000</td>
<td></td>
<td>53705</td>
<td>Y</td>
</tr>
</tbody>
</table>

### Bitmap index on “retweet”

<table>
<thead>
<tr>
<th>R-Yes</th>
<th>R-No</th>
<th>R-Null</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

- **SELECT COUNT(\(*)\) FROM Tweets WHERE retweet = ‘N’**

- **SELECT * FROM Tweets WHERE retweet IS NOT NULL**
Storing the Bitmap index

• One bitmap for each value, and one for Nulls
• Need to store each bitmap
• Simple method: 1 file for each bitmap
• Can compress the bitmap!

Index size?

When is a bitmap index more space efficient than a B+-tree?
(Re)consider the following table:

```sql
CREATE TABLE Tweets (  
  uniqueMsgID INTEGER,       -- unique message id  
  tstamp TIMESTAMP,     -- when was the tweet posted  
  uid INTEGER,       -- unique id of the user  
  msg VARCHAR (140), -- the actual message  
  zip INTEGER,       -- zipcode when posted  
  retweet BOOLEAN    -- retweeted?  
);
```

```sql
SELECT * FROM Tweets WHERE zip = 53706
```

Would we build a bitmap index on zipcode?
### Bit-sliced index

#### Table

<table>
<thead>
<tr>
<th>uniqueMsgID</th>
<th>...</th>
<th>zip</th>
<th>retweet</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>11324</td>
<td>Y</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>53705</td>
<td>Y</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>53706</td>
<td>N</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>53705</td>
<td>NULL</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>90210</td>
<td>N</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1,000,000,000,000</td>
<td></td>
<td>53705</td>
<td>Y</td>
</tr>
</tbody>
</table>

#### Query evaluation:
Walk through each slice constructing a **result bitmap**

E.g. zip $\leq 11324$, skip entries that have 1 in the first three slices (16, 15, 14)

---

#### Why do we have 17 bits for zipcode?

Bit-sliced index (1 slice per bit)

<table>
<thead>
<tr>
<th>Slice 16</th>
<th>Slice 1</th>
<th>Slice 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 1 0 1 1 0 0 0 0 0 1 1 1 1 0 0</td>
<td>0 1 1 0 1 0 0 0 1 1 1 0 0 1 0 0 1</td>
<td>0 1 1 0 1 0 0 0 1 1 1 0 0 1 0 1 0</td>
</tr>
<tr>
<td>0 1 1 0 1 0 0 0 1 1 1 0 0 1 0 0 1</td>
<td>0 1 1 0 1 0 0 0 1 1 1 0 0 1 0 1 0</td>
<td>1 0 1 1 0 0 0 0 0 0 1 1 0 0 0 1 0</td>
</tr>
</tbody>
</table>
| 0 1 1 0 1 0 0 0 1 1 1 0 0 1 0 0 1 | 0 1 1 0 1 0 0 0 1 1 1 0 0 1 0 1 0 | ...
| 0 1 1 0 1 0 0 0 1 1 1 0 0 1 0 0 1 | 0 1 1 0 1 0 0 0 1 1 1 0 0 1 0 1 0 | 0 0 0 0 0 0 0 0 1 1 0 0 0 1 0 |

---

Are we missing anything in the bit-sliced index above?
Bitslice Indices

• Can also do aggregates with Bitslice indices
  – E.g. SUM(attr): Add bit-slice by bit-slice.
    First, count the number of 1s in the slice17, and multiply the count by $2^{17}$
    Then, count the number of 1s in the slice16, and multiply the count by ...

• Store each slice using methods like what you have for a bitmap.
  – Note once again can use compression
Bitmap v/s Bitslice

• Bitmaps better for low cardinality domains
• Bitslice better for high cardinality domains
• Generally easier to “do the math” with bitmap indices