Compressed Data Structures for Query Processing

Introduction to Bitmap Indexes

Anna Bolotina

University of Salzburg

Advanced Topics in Data Analytics
Winter semester 2026/27

Possible change to class schedule

One option

Tuesdays, 8:45–11:00

Same room.

- Would this time work for everyone?
- What other times would work for everyone?

One bit per record

Record	Country
1	Austria
2	Germany
3	Austria
4	Italy
5	Austria
6	Germany

Predicate: country = 'Austria'

Record	1	2	3	4	5	6
Bit	1	0	?	?	?	?

Record 1: Austria, so its bit is **1**.

Record 2: Germany, so its bit is **0**.

Record 3: should its bit be 0 or 1?

A 1 means the record satisfies the predicate. A 0 means it does not.

Combining two conditions

AND rule

The result is 1 only when **both input bits are 1**.

Record	1	2	3	4	5	6
Austria	1	0	1	0	1	0
Premium	1	0	0	1	1	1
AND	**1**	**0**	?	?	?	?

Record 1: $1 \text{ AND } 1 = 1$. Record 2: $0 \text{ AND } 0 = 0$.

Record 3 has bits 1 and 0. What is its result bit?
Then complete records 4–6 and name the matching records.

Equality bitmap indexes

For attribute X and value v, define an N-bit vector:

$$B_{X=v}[i] = \begin{cases} 1, & \text{record } i \text{ has } X = v, \\ 0, & \text{otherwise.} \end{cases}$$

- Each position refers to one record.
- Each distinct attribute value has its own bitmap.
- A predicate selects the positions whose bits are 1.

Building an index from a column

Record	Country
1	Austria
2	Germany
3	Austria
4	Italy
5	Austria
6	Germany

Predicate	Bitmap
Austria	101010
Germany	010001
Italy	000100

Each row belongs to exactly one country bitmap.

The bitmap universe

Conditions for combining bitmaps

Same number of positions, same record order, same table snapshot.

- Position i must identify the same record in every input.
- In today's examples, each record has one non-NULL value per attribute.

Boolean operations on one position

a	b	a AND b	a OR b	NOT a
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

Bitwise evaluation

Apply the same rule independently at every record position.

Recap: Subscriber records

Record	Country	Subscription
1	Austria	Premium
2	Germany	Basic
3	Austria	Basic
4	Italy	Premium
5	Austria	Premium
6	Germany	Premium

AND: both predicates hold

Predicate	Records 1–6
Austria	101010
Premium	100111
Austria AND Premium	100010

$$\{1, 3, 5\} \cap \{1, 4, 5, 6\} = \{1, 5\}$$

Set interpretation

AND computes the intersection of matching records.

OR: at least one predicate holds

Predicate	Records 1–6
Austria	101010
Premium	100111
Austria OR Premium	101111

$$\{1, 3, 5\} \cup \{1, 4, 5, 6\} = \{1, 3, 4, 5, 6\}$$

Set interpretation

OR computes the union without duplicating records.

NOT: flipping each bit

Predicate	Records 1–6
Austria	101010
NOT Austria	010101

$$U = \{1, 2, 3, 4, 5, 6\}, \qquad U \setminus \{1, 3, 5\} = \{2, 4, 6\}$$

Bitwise NOT

NOT flips each bit: 0 becomes 1, and 1 becomes 0.
The result still has one bit per record.

Compound queries and parentheses

A compound predicate

```
country = 'Austria' AND
(subscription = 'Premium' OR subscription = 'Basic')
```

$$R = B_{\text{Austria}} \text{ AND } \left(B_{\text{Premium}} \text{ OR } B_{\text{Basic}}\right)$$

- Evaluate subexpressions and combine their results.
- Parentheses specify which conditions belong together.
- Every intermediate result is another bitmap over U.

U is the bitmap universe: the set of records represented by the bitmap positions. Here, $U = \{1, 2, 3, 4, 5, 6\}$.

Wordwise operations on raw bitmaps

A machine word groups several record positions

A 64-bit AND instruction combines 64 aligned bit pairs at once.

- A full result over N records needs $\lceil N/w \rceil$ words.
- A conventional full AND/OR scan costs $O(\lceil N/w \rceil)$.
- Reading inputs and writing output also consume bandwidth.

NOT over N valid bits

Only the N bits representing records belong to the result.
After wordwise NOT, mask out any padding bits in the last word.

w: bits per word. This describes a full scan with a materialized raw result.

Exercise: Boolean bitmap queries

Predicate	Records 1–6
A: Austria	101010
P: Premium	100111

1. Compute $A \,\mathrm{AND}\, (\mathrm{NOT} P)$ and list its record IDs.
2. Compute $(\mathrm{NOT} A) \,\mathrm{OR}\, P$.
3. Check $\mathrm{NOT}(A \,\mathrm{OR}\, P) = (\mathrm{NOT} A) \,\mathrm{AND}\, (\mathrm{NOT} P)$.

Universe: exactly six records.

A compound query: subscriber records

Requested records

Subscribers outside Italy who have Premium or are in Germany.

```
NOT (country = 'Italy')
AND (subscription = 'Premium' OR country = 'Germany')
```

Symbol	Predicate	Records 1–6
I	Country = Italy	000100
P	Subscription = Premium	100111
G	Country = Germany	010001

$$T_1 = \text{NOT}\, I, \quad T_2 = P\ \text{OR}\ G, \quad R = T_1\ \text{AND}\ T_2$$

Intermediate results: NOT and OR

Expression	Records 1–6
I (Italy)	000100
$T_1 = \mathrm{NOT}\, I$	111011
P (Premium)	100111
G (Germany)	010001
$T_2 = P \,\mathrm{OR}\, G$	110111

T_1: outside Italy.
T_2: Premium or Germany.

The final bitmap and matching records

Record	1	2	3	4	5	6
T_1	1	1	1	0	1	1
T_2	1	1	0	1	1	1
$R = T_1 \text{ AND } T_2$	**1**	**1**	0	0	**1**	**1**

Matching IDs: **1, 2, 5, 6**. $R = 110011$.

- Record 2 matches through Germany, although it has Basic.
- Record 3 fails the Premium-or-Germany condition.
- Record 4 fails the outside-Italy condition.

Break

10 minutes

Next: run-length encoding and queries over runs.

Run-length encoding

To study RLE, consider a bitmap with **12 positions**;
the underlying table is not needed here.

0000 111111 00

Read the sequence as **four zeros, six ones, two zeros**.

Group	Bit value	How many?	Pair
0000	0	4	$(0, 4)$
111111	1	6	$(1, 6)$
00	0	2	$(0, 2)$

A **run** is a maximal consecutive group of equal bits.
RLE stores one **(bit value, run length)** pair for each run.

Spaces show run boundaries; they are not extra positions.

Reading and completing RLE pairs

Reading a pair

$(1, 6)$ means **repeat the bit 1 six times**: `111111`.

The full example is $(0, 4), (1, 6), (0, 2)$.
Its lengths sum to $4 + 6 + 2 = 12$ positions.

Another sequence: `000 111 00`

Group	Bit value	Length	Pair
`000`	0	3	$(0, 3)$
`111`	1	3	$(1, 3)$
`00`	?	?	$(?, ?)$

What pair describes the final two bits?

Same density, different run counts

16-bit bitmap	Density	Runs
0000000011111111	50%	2
0011001100110011	50%	8
0101010101010101	50%	16

$$\text{density } p = \frac{\#\text{ones}}{N}, \qquad \text{mean run length} = \frac{N}{\#\text{runs}}$$

Clustering matters

RLE stores runs. Equal density can lead to different storage costs.

Sparsity/density, clustering, and RLE size

16-bit bitmap	Density	Runs
1000100010001000	25%	8
1111000000000000	25%	2
0000111111111111	75%	2

- Clustered ones create long one runs and long zero runs.
- Complementing a bitmap preserves its run lengths.
- Density alone does not determine RLE size.

An explicit storage model

Illustrative fixed-width pairs

Each run uses **17 bits**:
1 bit for the value + 16 bits for the length.

Run lengths in this model: 1–65,535.

For a 1,024-bit bitmap, **raw storage uses 1,024 bits**.

Bitmap pattern	Runs	RLE storage
512 zeros, then 512 ones	2	$2 \times 17 = 34$ bits
Alternating zero and one	1,024	$1{,}024 \times 17 = 17{,}408$ bits

Long runs can save space; alternating bits can increase storage.

Bit-packed payload only. Excludes representation metadata and allocation overhead.

Row order changes the runs

Order	Austria bitmap	Runs
Original IDs 1–6	101010	6
IDs 1, 3, 5, 2, 6, 4	111000	2

- Grouping equal country values creates longer country runs.
- Apply the same row permutation to every bitmap.
- Keep a mapping from new positions to original record IDs.
- One 32-bit ID per position $= 4$ bytes per position.
  For 6 records: $6 \times 4 = 24$ bytes.
- The mapping is shared by all bitmaps in the same row order.
- Sorting one attribute may give little benefit for another.

Lemire, Kaser and Aouiche: Sorting improves word-aligned bitmap indexes

Sorting groups the matching positions

Constructed example: 8,192 records; one attribute with 64 distinct values; one equality bitmap per value.

Before sorting: chosen row order

$\underbrace{1, 2, \ldots, 64}_{\text{64 records}} \mid \underbrace{1, 2, \ldots, 64}_{\text{64 records}} \mid \cdots$

128 repetitions of the same sequence.

Chosen to separate equal values. Other orders are possible.

$128 \times 64 = 8{,}192$ records.

Equality bitmaps before sorting

Bitmap for value 1: $\underbrace{1, 0, \ldots, 0}_{\text{64 bits}} \mid \cdots$

Bitmap for value 2: $\underbrace{0, 1, \ldots, 0}_{\text{64 bits}} \mid \cdots$

...

Each raw bitmap has 128 blocks of 64 bits = 8,192 bits.

After sorting: 128 records in each group

$\underbrace{1, \ldots, 1}_{\text{128 records}} \mid \underbrace{2, \ldots, 2}_{\text{128 records}} \mid \cdots \mid \underbrace{64, \ldots, 64}_{\text{128 records}}$

Bitmap for value 1: **128 ones, then 8,064 zeros**.

Same records and predicates; fewer runs in every equality bitmap.

Does sorting pay for the mapping?

Same example: 8,192 records, 64 equality bitmaps; 32-bit original IDs.

Across all 64 bitmaps	Before sorting	After sorting
Runs	$2 \times 256 + 62 \times 257 = 16{,}446$	$2 \times 2 + 62 \times 3 = 190$
RLE payload (17 bits/run)	$16{,}446 \times 17 = 279{,}582$ bits	$190 \times 17 = 3{,}230$ bits
Shared mapping (32-bit IDs)	—	$8{,}192 \times 32 = 262{,}144$ bits
Total	**279,582 bits**	**265,374 bits**

Net saving: $279{,}582 - 265{,}374 = 14{,}208$ bits $=$ **1,776 bytes**.

Raw bitmaps: $64 \times 8{,}192 = 524{,}288$ bits $= 65{,}536$ bytes.

Same bit-packed RLE model; excludes metadata and byte-alignment overhead.

Exercise: density and RLE size

Name	16 bits
C	0000000000001111
D	0001000100010001
E	1111111111110000

1. Find the density and write the maximal RLE pairs.
2. Compare run counts. Which pair of bitmaps has equal density?
3. Under the 17-bit pair model, does any beat 16 raw bits?

RLE exercise: solution

Bitmap	Density	Runs	RLE
C	25%	2	$(0, 12), (1, 4)$
D	25%	8	$[(0, 3), (1, 1)]$ repeated 4 times
E	75%	2	$(1, 12), (0, 4)$

Overhead at small scale

Two pairs cost 34 bits. Eight pairs cost 136 bits.
Each exceeds the 16-bit raw payload in this model.

The repeated-pair notation describes eight stored pairs. RLE does not encode that repetition separately.

Boolean operations on runs

A and B are the input bitmaps; a and b are their current run values.

Safe processing length

$t = \min(\text{bits remaining in A's current run},\ \text{bits remaining in B's current run})$.

Process t positions from each current run in one step.
The result has the same bit value throughout these positions.
For each input bitmap separately:
if its current run ends, move to the next run; otherwise stay in the current run.

a	b	Operation	Output run
0	any	AND	$(0, t)$
1	b	AND	(b, t)
1	any	OR	$(1, t)$
0	b	OR	(b, t)

Boolean operations on runs

A and B are the input bitmaps; a and b are their current run values.

Safe processing length

$t = \min(\text{bits remaining in A's current run},\ \text{bits remaining in B's current run})$.

Process t positions from each current run in one step.
The result has the same bit value throughout these positions.
For each input bitmap separately:
if its current run ends, move to the next run; otherwise stay in the current run.

a	b	Operation	Output run
0	any	AND	$(0, t)$
1	b	AND	(b, t)
1	any	OR	$(1, t)$
0	b	OR	(b, t)

Example: first step of computing A AND B

`A: 0000 | 111111 | 00`
`B: 00 | 1111 | 0000 | 11`
Step 1: process the first two positions (red).
$t = \min(4, 2) = 2$; output: $(0, 2)$.
A stays in its current run: 2 zeros left.
B moves to its next run.

Full example: Run-based computation of A AND B

```
A: 0000 | 111111 | 00
B: 00 | 1111 | 0000 | 11
```

Positions	a	b	t	Emitted segment
1–2	0	0	2	$(0, 2)$
3–4	0	1	2	$(0, 2)$
5–6	1	1	2	$(1, 2)$
7–10	1	0	4	$(0, 4)$
11–12	0	1	2	$(0, 2)$

Same bits, split by processing step

```
A:   00 | 00 | 11 | 1111 | 00
B:   00 | 11 | 11 | 0000 | 11
AND: 00 | 00 | 11 | 0000 | 00
```

Result after merging adjacent equal segments

$(0, 4), (1, 2), (0, 6)$ Check: `000011000000`.

Five interval steps compute the 12-position result.

NOT and counting directly on runs

NOT changes the values

A : $(0,4),(1,6),(0,2)$

$\mathrm{NOT}A$: $(1,4),(0,6),(1,2)$

Run lengths stay the same.

COUNT sums one-run lengths

$A \mathrm{AND} B$: $(0,4),(1,2),(0,6)$

Matching record count: 2.

A count-only query can sum lengths as the merge proceeds.

NOT flips the bits at all N valid positions; the bitmap length stays N.

Benefits and limits of queries over runs

Method	Work for a full binary merge	Effect of bit pattern
Raw bitmap	$O(\lceil N/w \rceil)$ word operations	None
RLE pair streams	$O(r_A + r_B)$ run steps	Fewer runs mean fewer processing steps

- Long runs can reduce input reads and intermediate storage.
- Alternating bits cause many runs and extra bookkeeping.
- Random access needs scanning or an auxiliary index.
- Returning many IDs or rows still costs time.

Run steps and word operations have different costs. Measure both space and query time.

What the examples establish

- Each bitmap bit indicates whether the record at that position satisfies the predicate.
- Boolean expressions combine membership sets.
- In our fixed-width model, RLE size depends on the number of runs.
- RLE queries combine aligned intervals without full expansion.

Next session

WAH, EWAH and Roaring: representations and trade-offs.
Assignment 1: Compression Benchmark.

Discussion and questions

1. Can two bitmaps with the same density have very different RLE sizes?
2. Why do all bitmap indexes need the same row order?
3. When might a raw bitmap outperform RLE?
4. How does counting matches differ from returning all matching IDs?

What would you vary in a compression benchmark?

Boolean exercise: answers

Expression	Positions 1–6
$\mathrm{NOT}\,P$	011000
$A\ \mathrm{AND}\ (\mathrm{NOT}\,P)$	001000
$(\mathrm{NOT}\,A)\ \mathrm{OR}\ P$	110111
$\mathrm{NOT}(A\ \mathrm{OR}\ P)$	010000
$(\mathrm{NOT}\,A)\ \mathrm{AND}\ (\mathrm{NOT}\,P)$	010000

Matching IDs: $\{3\}$, $\{1, 2, 4, 5, 6\}$, and $\{2\}$ respectively.

Extra exercise: AND over RLE

Input	Expanded view for checking
$A = (0,4),(1,4),(0,4),(1,4)$	0000111100001111
$B = (0,2),(1,4),(0,4),(1,4),(0,2)$	0011110000111100

1. Compute $A \,\mathrm{AND}\, B$ step by step using the input runs.
2. Merge adjacent output segments with the same bit value, then count the matching records.

Optional practice. Use the pair streams as inputs.

SQL NULLs and a valid-record universe

A NULL is an unknown value

SQL `WHERE country <> 'Austria'` excludes NULL countries. Blindly complementing the Austria bitmap would include them.

$$R = U \text{ AND } (\text{NOT} B_{\text{Austria}}) \text{ AND } (\text{NOT} B_{\text{country IS NULL}})$$

- U marks valid records in the query's snapshot.
- A NULL bitmap excludes unknown country values here.
- General SQL expressions need three-valued logic handling.

Optional reading for the next session

- K. Wu, E. J. Otoo and A. Shoshani.
  Optimizing Bitmap Indices With Efficient Compression.
  ACM Transactions on Database Systems 31(1), 2006.
  Author-hosted paper and publication details
- D. Lemire, O. Kaser and K. Aouiche.
  Sorting improves word-aligned bitmap indexes.
  Data & Knowledge Engineering 69(1), 2010.
  Paper on arXiv
- FastBit: Word-Aligned Hybrid Compression for Bitmap Indices