Database Cracking

David Werner
January 23, 2018

Technische Universität München
Database cracking
What is database cracking? - 1

• self organized indexing and index maintenance
• queries are used as advice to crack the database in pieces
• cracking means physically reordering the database
• sequential access for range queries is guaranteed
What is database cracking? - 2

- original column stays in insertion order
- cracking column is used for reordering
- this allows fast reconstruction of records
Cracking example

Query 1: $X < 5$

Query 2: $5 \leq X \leq 8$

Query 2: $8 < X$
• index on cracking column
• stores information about every crack
  • bound value
  • end position of piece
  • inclusive flag
Cracking in two pieces - basic

Algorithm 1 Crack in two pieces

```plaintext
1: procedure CRACK_IN_2(column, left, right, value, inclusive)
2:     while left < right do
3:         if column[left] \( \Delta_1 \) value then
4:             left ← left + 1
5:         else
6:             while column[right] \( \Delta_2 \) value and left < right do
7:                 right ← right − 1
8:             end while
9:             swap(column[left], column[right])
10:            left ← left + 1
11:            right ← right − 1
12:        end if
13:     end while
14: end procedure
```

\( \Delta_1 \) is \(< \) or \( \leq \), \( \Delta_2 \) is \(> \) or \( \geq \) depending on the inclusive flag
Algorithm 2 Crack in two pieces (branch free)

1: procedure CRACK_IN_2_BF(column, left, right, value, inclusive)
2:   cmp
3:   active ← column[left]
4:   backup ← column[right]
5:   while left < right do
6:       cmp ← active \(\Delta_1\) value
7:       column[left] ← active
8:       column[right] ← active
9:       left ← left + cmp
10:      right ← right − (1 − cmp)
11:     active ← (column[left] \* cmp) + (column[right] \* (1 − cmp))
12:   swap(active, backup)
13: end while
14: column[left] ← active
15: end procedure
Algorithm 3 Crack in three pieces

1: procedure CRACK_IN_3(column, left, right, value1, value2, inclusive1, inclusive2)
2:     tmp ← left
3:     while left < right do
4:         while left < right and column[left] Δ1 value2 do
5:             if column[left] Δ1 value1 then
6:                 swap(column[left], column[tmp])
7:                 tmp ← tmp + 1
8:         end if
9:         left ← left + 1
10:     end while
11:     while left < right and column[right] Δ2 value2 do
12:         right ← right − 1
13:     end while
14:     if left < right then
15:         swap(column[left], column[right])
16:     end if
17: end procedure
Advantages

Database cracking has some interesting properties:

- no copying of query results
- no upfront knowledge about workload required
- physical reordering can be supported by index
- consecutive cracks receive speed from index
Implementation
Cracking Algorithms

- All three cracking algorithms
- Return last position of piece in cracking column
- \(<\) and \(\leq\) cracks only
- \(>\) and \(\geq\) queries can use these results
Cracking index struct

• Combines cracking algorithms with cracking index

• Comprises:
  • Pointer to original column
  • Pointer to cracking column
  • Column size
  • Map as index

• Main functionality:
  • Find pieces
  • Query (single bound, double bound)
exact match:

returns: true
no match at all\(^1\) or inclusive flag does not match\(^2\): 

returns: false
Two different types of queries

- single bound (e.g. $X < a$)
- double bound (e.g. $a < X < b$)

Query method interface:

- Require bound value(s) and inclusive flag(s)
- Return start/end position of result piece(s)
simple control flow:

1. Find piece for value
2. If exact match: return
3. Otherwise: crack
4. Add crack to index
5. Return
Query - double bound

- Find piece for both bounds
- Depending on results different cases need to be handled
- Four easy cases:
  - None of both bounds needs a crack
  - Both bounds need crack in different pieces
  - Upper/lower bound needs crack
- Two involved cases
Special case 1

example query: $9 \leq X < 12$

solution: crack in three pieces
Special case 2

example query: $4 \leq X \leq 13$

solution: crack yellow first, use result to crack red
Extensions and Usage

- **Extensions:**
  - Leaves have sibling pointers
  - Pointer to leftmost leaf

- **Tree stores:**
  - bound values as keys
  - position and inclusive flag as payload
Query operation

1. Find start position
2. Find end position
3. Traverse leaves
4. Lookup column positions
5. Copy column values to output
6. Stop at end position
Evaluation
Test cases

• Comparison of „Crack in two“ algorithms
  • 500'000'000
  • single crack

• Cracking vs. Indexing
  • 50’000’000 values in column
  • 100 consecutive cracks
Comparison of cracking algorithms

small result piece:

big result piece:
Cracking vs Indexing

single crack workload:

only cracks workload: