>>

SIMC Indexing

∧ >>

❖ Signature-based indexing

Reminder: file organisation for signature indexing (two files)

One signature slot per tuple slot; unused signature slots are zeroed.

❖ Superimposed Codewords (SIMC)

In a superimposed codewords (simc) indexing scheme

- a tuple descriptor is formed by overlaying attribute codewords
- each codeword is *m* bits long and has *k* bits set to 1

A tuple descriptor *desc(t)* is

- a bit-string, *m* bits long, where $j \leqslant nk$ bits are set to 1
- $desc(t) = cw(A_1)$ **OR** $cw(A_2)$ **OR** ... **OR** $cw(A_n)$

Method (assuming all *n* attributes are used in descriptor):

```
bits desc = 0
for (i = 1; i <= n; i++) {
   bits cw = codeword(A[i],m,k)
   desc = desc | cw
}
```

COMP9315 21T1 ♢ SIMC Indexing ♢ [2/16]

❖ SIMC Example

Consider the following tuple (from bank deposit database)

Branch	AcctNo	Name	Amount
Perryridge	102	Hayes	400

It has the following codewords/descriptor (for $m = 12,\ \ k = 2$)

A_i	$cw(A_i)$
Perryridge	**010000000001**
102	**000000000011**
Hayes	**000001000100**
400	**000010000100**
$desc(t)$	**010011000111**

<< ∧ >>

❖ SIMC Queries

To answer query *q* in SIMC

- first generate *desc(q)* by OR-ing codewords for known attributes
- then attempt to match *desc(q)* against all signatures in sig file

E.g. consider the query **`(Perryridge, ?, ?, ?)`**.

A_i	$cw(A_i)$
Perryridge	**`010000000001`**
`?`	**`000000000000`**
`?`	**`000000000000`**
`?`	**`000000000000`**
desc(q)	**`010000000001`**

<< ^ >>

❖ SIMC Queries (cont)

Once we have a query descriptor, we search the signature file:

```
pagesToCheck = {}
// scan r signatures
for each descriptor D[i] in signature file {
    if (matches(D[i],desc(q))) {
        pid = pageOf(tupleID(i))
        pagesToCheck = pagesToCheck ∪ pid
    }
}
// then scan bsq = bq + δ pages to check for matches
```

Matching can be implemented efficiently ...

```
#define matches(sig,qdesc) ((sig & qdesc) == qdesc)
```

<< ^ >>

❖ Example SIMC Query

Consider the query and the example database:

Signature	Deposit Record
010000000001	(Perryridge,?,?,?)
100101001001	(Brighton,217,Green,750)
010011000111	(Perryridge,102,Hayes,400)
101001001001	(Downtown,101,Johnshon,512)
101100000011	(Mianus,215,Smith,700)
010101010101	(Clearview,117,Throggs,295)
100101010011	(Redwood,222,Lindsay,695)

Gives two matches: one true match, one false match.

❖ SIMC Parameters

False match probablity p_F = likelihood of a false match

How to reduce likelihood of false matches?

- use different hash function for each attribute (h_i for A_i)
- increase descriptor size (m)
- choose k so that ≅ half of bits are set

Larger m means larger signature file ⇒ read more signature data.

Having k too high ⇒ increased overlapping.
Having k too low ⇒ increased hash collisions.

<< ∧ >>

❖ SIMC Parameters (cont)

How to determine "optimal" m and k?

1. start by choosing acceptable p_F
   (e.g. $p_F \leqslant 10^{-4}$ i.e. one false match in 10,000)
2. then choose m and k to achieve no more than this p_F.

Formulae to derive m and k given p_F and n:

$$k = 1/\log_e 2 \, . \, \log_e (1/p_F)$$

$$m = (1/\log_e 2)^2 \, . \, n \, . \, \log_e (1/p_F)$$

Formula from Bloom (1970), "Space/Time Trade-offs in Hash Coding with Allowable Errors", Communications of the ACM, 13 (7): 422–426

❖ Query Cost for SIMC

Cost to answer *pmr* query: $Cost_{pmr} = b_D + b_{sq}$

- read r descriptors on b_D descriptor pages
- then read b_{sq} data pages and check for matches

$b_D = ceil(r/c_D)$ and $c_D = floor(B/ceil(m/8))$

E.g. $m=64, \ B=8192, \ r=10^4 \ \Rightarrow \ c_D = 1024, \ b_D=10$

b_{sq} includes pages with r_q matching tuples and r_F false matches

Expected false matches = $r_F = (r - r_q).p_F \cong r.p_F$ if $r_q \ll r$

E.g. Worst $b_{sq} = r_q+r_F$, Best $b_{sq} = 1$, Avg $b_{sq} = ceil(b(r_q+r_F)/r)$

❖ Page-level SIMC

SIMC has one descriptor per tuple ... potentially inefficient.

Alternative approach: one descriptor for each data page.

Every attribute of every tuple in page contributes to descriptor.

Size of page descriptor (PD) (clearly larger than tuple descriptor):

- use above formulae but with *c.n* "attributes"

E.g. $n = 4, c = 64, p_F = 10^{-3} \Rightarrow m_p \cong 3680 bits \cong 460 bytes$

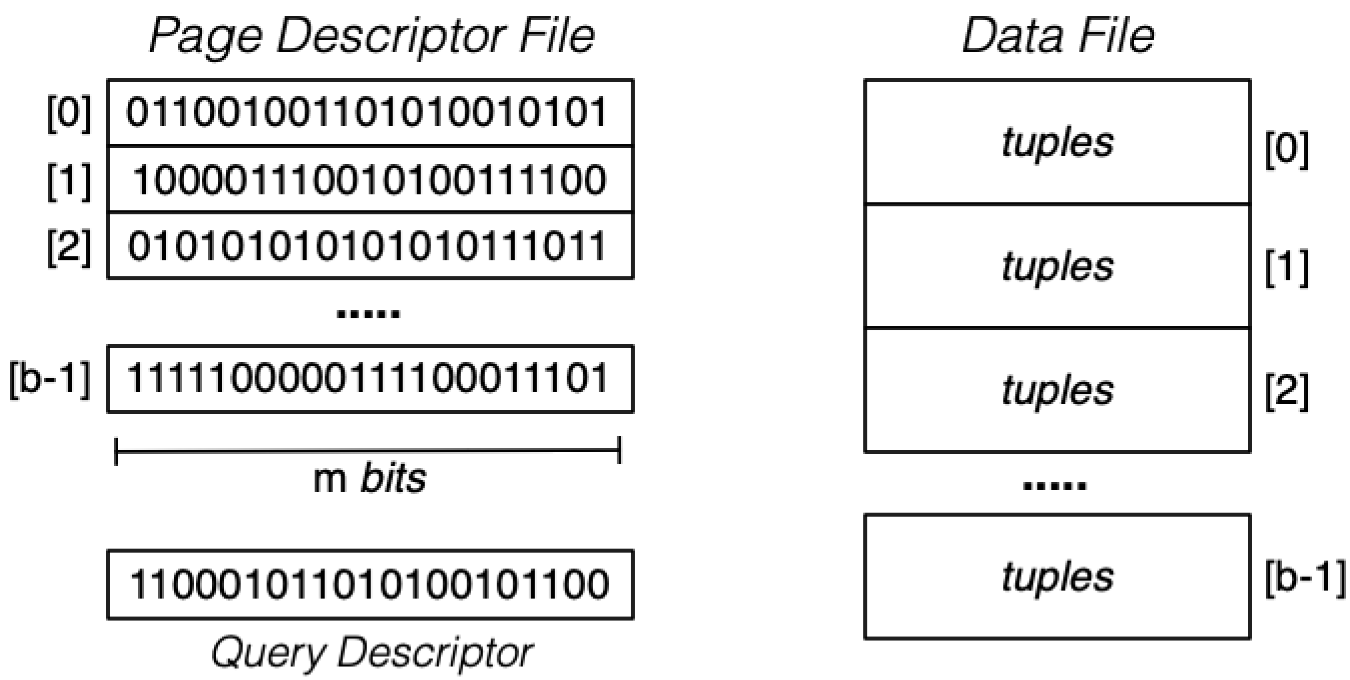
Typically, pages are 1..8KB $\Rightarrow$ 8..64 PD/page (c_{PD}).

E.g. $m_p \cong 460, \ \ B = 8192, \ \ c_{PD} \cong 17$

<< ^ >>

❖ Page-level SIMC (cont)

File organisation for page-level superimposed codeword index



<< ^ >>

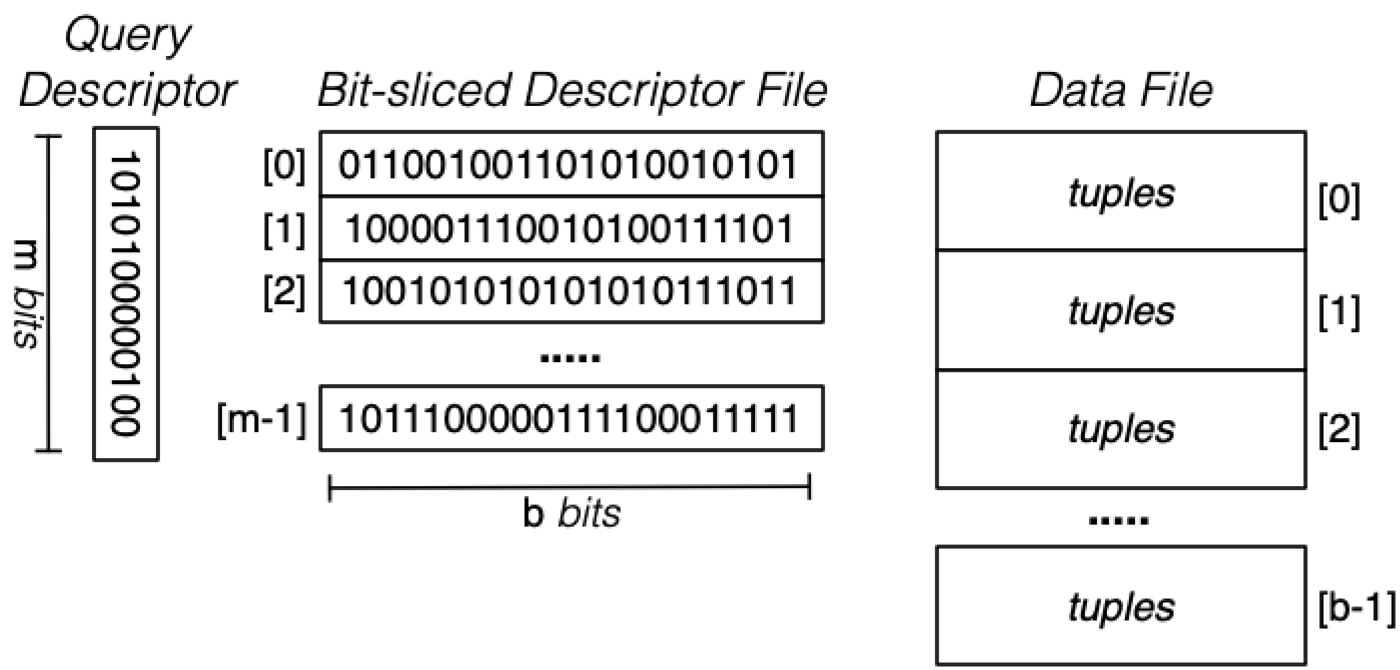
❖ Page-level SIMC (cont)

Algorithm for evaluating *pmr* query using page descriptors

```
pagesToCheck = {}
// scan b mp-bit page descriptors
for each descriptor D[i] in signature file {
    if (matches(D[i],desc(q))) {
        pid = i
        pagesToCheck = pagesToCheck ∪ pid
    }
}
// read and scan bsq data pages
for each pid in pagesToCheck {
    Buf = getPage(dataFile,pid)
    check tuples in Buf for answers
}
```

❖ Bit-sliced SIMC

Improvement: store *b* *m*-bit page descriptors as *m* *b*-bit "bit-slices"



<< ^ >>

❖ Bit-sliced SIMC (cont)

Algorithm for evaluating *pmr* query using bit-sliced descriptors

```
matches = ~0   //all ones
// scan m r-bit slices
for each bit i set to 1 in desc(q) {
    slice = fetch bit-slice i
    matches = matches & slice
}
for each bit i set to 1 in matches {
    fetch page i
    scan page for matching records
}
```

Effective because *desc(q)* typically has less than half bits set to 1

❖ Comparison of Approaches

Tuple-based

- r signatures, m-bit signatures, k bits/attribute
- read all pages of signature file in filtering for a query

Page-based

- b signatures, m_p-bit signatures, k bits/attribute
- read all pages of signature file in filtering for a query

Bit-sliced

- m signatures, b-bit slices, k bits/attribute
- read less than half of the signature file in filtering for a query

All signature files are roughly the same size, for a given p_F

❖ Signature-based Indexing in PostgreSQL

PostgreSQL supports signature based indexing via the **`bloom`** module

(Signature-based indexes like this are often called Bloom filters)

Creating a Bloom index

```
create index Idx on R using bloom (a1,a2,a3)
with   (length=64, col1=3, col2=4, col3=2);
```

Example: 10000 tuples, query = select * from R where a1=55 and a2=42, no matching tuples, random numeric values for attributes

```
No indexes ... execution time 15ms
B-tree index on all attributes ... execution time 12ms
Bloom index ... execution time 0.4ms
```

For more details, see PostgreSQL doc, Appendix F.5

Produced: 6 Apr 2021