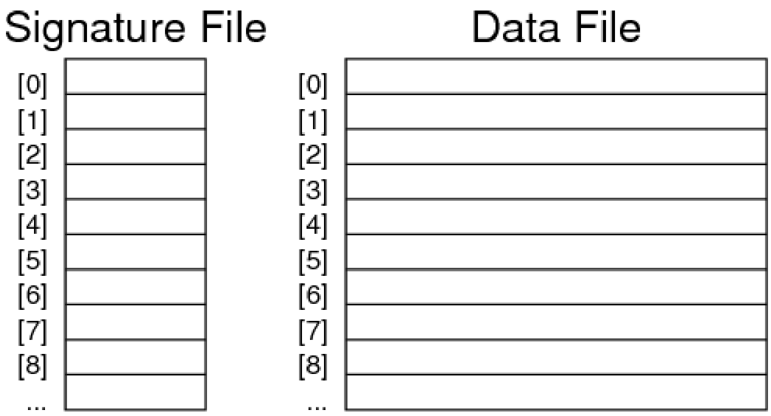
>>

CATC Indexing

❖ Signature-based indexing

Reminder: file organisation for signature indexing (two files)



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

We use the terms "signature" and "descriptor" interchangeably

<< ^ >>

❖ Concatenated Codewords (CATC)

In a concatenated codewords (catc) indexing scheme

- a tuple signature is formed by concatenating attribute codewords
- the signature is m bits long, with $\cong m/2$ bits set to 1
- codeword for $attr_i$ is u_i bits long and has $\cong u_i/2$ bits set to 1
- each codeword could be different length, but always $\Sigma_{1..n}\, u_i = m$

A tuple descriptor (signature) *desc(t)* is

- $desc(t) = cw(A_n) + cw(A_{n-1}) ... + cw(A_+) + cw(A_1)$
- where "+" represents bit-string concatentation

The order that the concenated codewords appears doesn't matter, as long as it's done consistently

❖ CATC Example

Consider the following tuple (from bank deposit database)

Branch	AcctNo	Name	Amount
Perryridge	102	Hayes	400

It has the following codewords/descriptor (for $m = 16,\ \ u_i = 4$)

A_i	$cw(A_i)$
Perryridge	0101
102	1001
Hayes	1010
400	1100
$desc(t)$	1100101010010101

❖ CATC Queries

To answer query *q* in CATC

- first generate *desc(q)* by combining codewords for all attributes
- for known A_i use $cw(A_i)$; for "unknown" A_i, use $cw(A_i) = 0$

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

A_i	$cw(A_i)$
Perryridge	**`0101`**
`?`	**`0000`**
`Hayes`	**`1010`**
`?`	**`0000`**
desc(q)	**`0000101000000101`**

<< ∧ >>

❖ CATC 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 U pid
    }
}
// then scan b_sq = b_q + 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
0000101000000101	(Perryridge,?,Hayes,?)
1010100101101001	(Brighton,217,Green,750)
1100101010010101	(Perryridge,102,Hayes,400)
1010011010010110	(Downtown,101,Johnshon,512)
0110101001010011	(Mianus,215,Smith,700)
1010101011000101	(Clearview,117,Throggs,295)
1001010100111001	(Redwood,222,Lindsay,695)

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

<< ∧ >>

❖ CATC 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)

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

Since u_i's are relatively small, hash collisions may be a serious issue

But making u_i's means larger signatures ⇒ optimisation problem

❖ CATC Parameters (cont)

How to determine "optimal" m and u?

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

Formulae to derive "good" m: $m = (1/\log_e 2)^2 . n . \log_e(1/p_F)$

Choice of u_i values

- each A_i has same u_i, or
- allocate u_i based on size of attribute domains

<< ^ >>

❖ Query Cost for CATC

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 $b_{sq} = ceil(b(r_q+r_F)/r)$

❖ Variations on CATC

CATC 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 $m_p = (1/\log_e 2)^2 . c.n . \log_e (1/p_F)$

Size of codewords is proportionally larger (unless attribute domain small)

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

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

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

❖ Variations on CATC (cont)

Improvement: store $b \times m_p$-bit page descriptors as $m_p \times b$-bit "bit-slices"

If $b = 2^x$ then uses same storage as page descriptors

Query cost: scan $u_i / 2$ bit-slices for each known attribute

If k is set of known attribute values, #slices = $\sum_{i \in k} u_i / 2$

E.g. $b = 128$, $m = 256$, $n = 4$, $u_i = 16$

`(a,?,c,?)` requires scan of 2×8 128-bit (16-byte) slices

compared to scan of 128 page descriptors, where each PD is 64-bits (8-bytes)

<< ^

❖ Comparison with SIMC

Assume same m, p_F, n for each method ...

CATC has u_i-bit codewords, each has $\cong u_i / 2$ bits set to 1

SIMC has m-bit codewords, each has k bits set to 1

Signatures for both have m bits, with $\cong m / 2$ bits set to 1

CATC has flexibility in u_i, but small(er) codewords so more hash collisions

SIMC has less hash collisions, but has errors from "unfortunate" overlays

Produced: 28 Mar 2021