>>

Hashing in PostgreSQL

^ >>

❖ Hashing in PostgreSQL

PostgreSQL uses linear hashing on tables which have been:

```
create index Ix on R using hash (k);
```

Hash file implementation: **backend/access/hash**

- **hashfunc.c** ... a family of hash functions
- **hashinsert.c** ... insert, with overflows
- **hashpage.c** ... utilities + splitting
- **hashsearch.c** ... iterator for hash files

Detailed info in **src/backend/access/hash/README**

Based on "A New Hashing Package for Unix", Margo Seltzer, Winter Usenix 1991

<< ^ >>

❖ PostgreSQL Hash Function

PostgreSQL generic hash function (simplified):

```
Datum hash_any(unsigned char *k, int keylen)
{
   uint32 a, b, c, len, *ka = (uint32 *)k;
   /* Set up the internal state */
   len = keylen;
   a = b = c = 0x9e3779b9+len+3923095;
   /* handle most of the key */
   while (len >= 12) {
      a += ka[0]; b += ka[1]; c += ka[2];
      mix(a, b, c);
      ka += 3;  len -= 12;
   }
   ... collect data from remaining bytes into a,b,c ...
   mix(a, b, c);
   return UInt32GetDatum(c);
}
```

See **backend/access/hash/hashfunc.c** for details (incl **mix()**)

<< ^ >>

❖ PostgreSQL Hash Function (cont)

`hash_any()` gives hash value as 32-bit quantity (**`uint32`**).

Typically invoked from a type-specific function, e.g.

```
Datum
hashint4(PG_FUNCTION_ARGS)
{
    return hash_uint32(PG_GETARG_INT32(0));
}
```

where **`hash_uint32()`** is a faster version of **`hash_any()`**

Hash value is "wrapped" as a **`Datum`**

❖ PostgreSQL Hash Function (cont)

Implementation of hash → page ID

```
Bucket
_hash_hashkey2bucket(uint32 hashkey, uint32 maxbucket,
                     uint32 highmask, uint32 lowmask)
{
    Bucket      bucket;

    bucket = hashkey & highmask;
    if (bucket > maxbucket)
        bucket = bucket & lowmask;

    return bucket;
}
```

<< ^ >>

❖ Hash Files in PostgreSQL

PostgreSQL uses different file organisation ...

- has a single file containing header, main and overflow pages
- has groups of main pages of size 2^n
- in between groups, arbitrary number of overflow pages
- maintains collection of group pointers in header page
- each group pointer indicates start of main page group

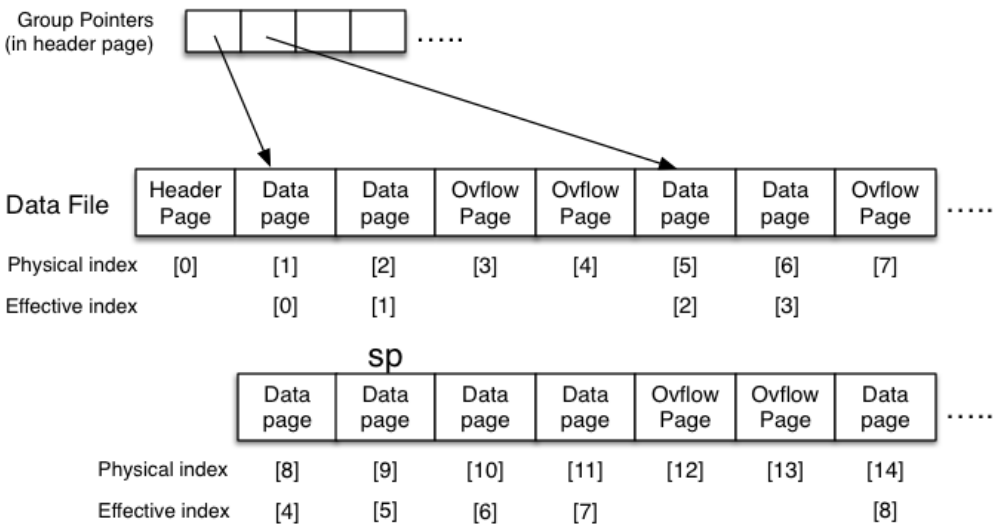
If overflow pages become empty, add to free list and re-use.

Confusingly, PostgreSQL calls "group pointers" as "split pointers"

<< ^ >>

❖ Hash Files in PostgreSQL (cont)

PostgreSQL hash file structure:



COMP9315 21T1 ♢ PostgreSQL Hashing ♢ [6/7]

<< ^

❖ Hash Files in PostgreSQL (cont)

Approximate method for converting bucket # to page address:

```
// which page is primary page of bucket
uint bucket_to_page(headerp, B) {
    uint *splits = headerp->hashm_spares;
    uint chunk, base, offset, lg2(uint);
    chunk = (B<2) ? 0 : lg2(B+1)-1;
    base = splits[chunk];
    offset = (B<2) ? B : B-(1<<chunk);
    return (base + offset);
}
// returns ceil(log_2(n))
int lg2(uint n) {
    int i, v;
    for (i = 0, v = 1; v < n; v <<= 1) i++;
    return i;
}
```

Produced: 11 Mar 2021