>>

# Heap Files

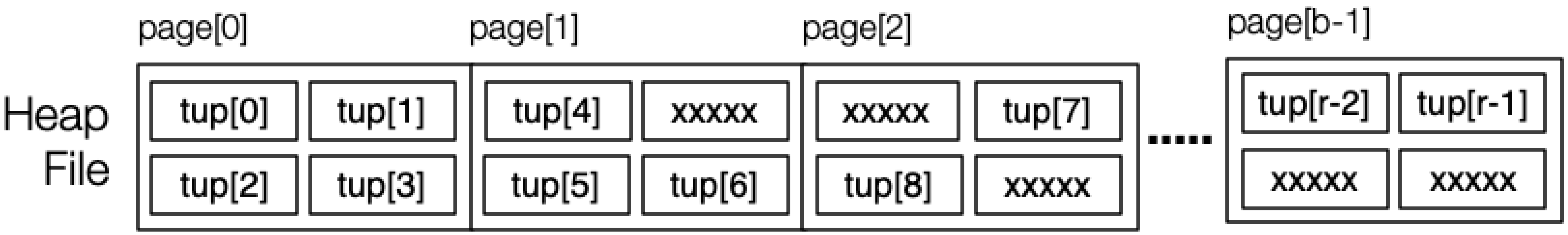
- Heap Files
- Selection in Heaps
- Insertion in Heaps
- Deletion in Heaps
- Updates in Heaps
- Heaps in PostgreSQL

/\ >>

# ❖ Heap Files

Heap files

- sequence of pages containing tuples
- no inherent ordering of tuples (added in next free slot)
- pages may contain free space from deleted tuples
- does not generally involve overflow pages



#pages = b, #tuples = r, capacity c=4

Note: this is **not** "heap" as in the top-to-bottom ordered tree.

<< ^ >>

## ❖ Selection in Heaps

For all selection queries, the only possible strategy is:

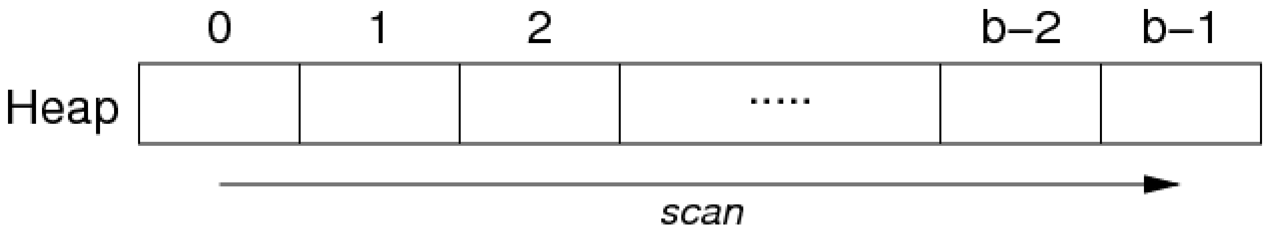
```
// select * from R where C
rel = openRelation("R", READ);
for (p = 0; p < nPages(rel); p++) {
    get_page(rel, p, buf);
    for (i = 0; i < nTuples(buf); i++) {
        T = get_tuple(buf, i);
        if (T satisfies C)
            add tuple T to result set
    }
}
```

i.e. linear scan through file searching for matching tuples

<< ^ >>

# ❖ Selection in Heaps (cont)

The heap is scanned from the first to the last page:



$Cost_{range} = Cost_{pmr} = b$

If we know that only one tuple matches the query (*one* query),
a simple optimisation is to stop the scan once that tuple is found.

$Cost_{one}$ : Best = *1* Average = *b/2* Worst = *b*

<< ∧ >>

## ❖ Insertion in Heaps

Insertion: new tuple is appended to file (in last page).

```
rel = openRelation("R", READ|WRITE);
pid = nPages(rel)-1;
get_page(rel, pid, buf);
if (size(newTup) > size(buf))
   { deal with oversize tuple }
else {
   if (!hasSpace(buf,newTup))
      { pid++; nPages(rel)++; clear(buf); }
   insert_record(buf,newTup);
   put_page(rel, pid, buf);
}
```

$$Cost_{insert} = 1_r + 1_w$$

<< ^ >>

## ❖ Insertion in Heaps (cont)

Alternative strategy:

- find any page from `R` with enough space
- preferably a page already loaded into memory buffer

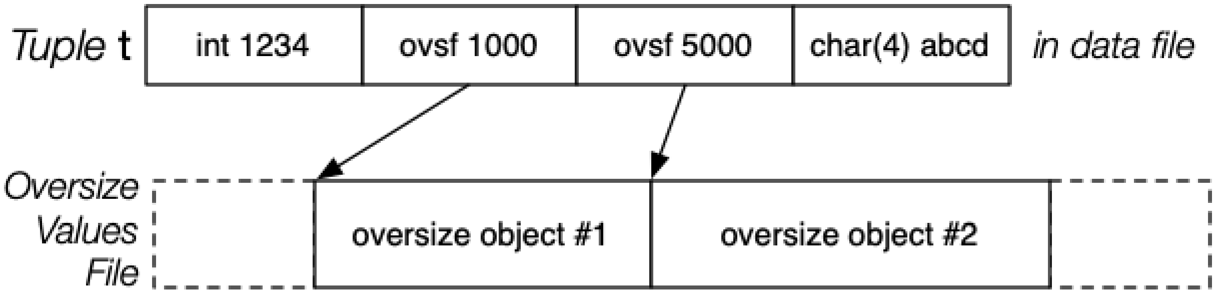
PostgreSQL's strategy:

- use last updated page of `R` in buffer pool
- otherwise, search buffer pool for page with enough space
- assisted by free space map (FSM) associated with each table
- for details: `backend/access/heap/{heapam.c,hio.c}`

# ❖ Insertion in Heaps (cont)

Dealing with oversize tuple **t**:

```
for i in 1 .. nAttr(t) {
   if (t[i] not oversized) continue
   off = appendToFile(ovf, t[i])
   t[i] = (OVERSIZE, off)
}
insert into buf as before
```



<< ^ >>

## ❖ Insertion in Heaps (cont)

PostgreSQL's tuple insertion:

```
heap_insert(Relation relation,    // relation desc
            HeapTuple newtup,     // new tuple data
            CommandId cid, ...)   // SQL statement
```

- finds page which has enough free space for **`newtup`**
- ensures page loaded into buffer pool and locked
- copies tuple data into page buffer, sets **`xmin`**, etc.
- marks buffer as dirty
- writes details of insertion into transaction log
- returns OID of new tuple if relation has OIDs

<< ∧ >>

## ❖ Deletion in Heaps

SQL: **`delete from`** *R* **`where`** *Condition*

Implementation of deletion:

```
rel = openRelation("R",READ|WRITE);
for (p = 0; p < nPages(rel); p++) {
    get_page(rel, p, buf);
    ndels = 0;
    for (i = 0; i < nTuples(buf); i++) {
        tup = get_tuple(buf,i);
        if (tup satisfies Condition)
            { ndels++; delete_record(buf,i); }
    }
    if (ndels > 0) put_page(rel, p, buf);
    if (ndels > 0 && unique) break;
}
```

<< ∧ >>

## ❖ Deletion in Heaps (cont)

PostgreSQL tuple deletion:

```
heap_delete(Relation relation,      // relation desc
            ItemPointer tid, ...,  // tupleID
            CommandId cid, ...)    // SQL statement
```

- gets page containing tuple **`tid`** into buffer pool and locks it
- sets flags, commandID and **`xmax`** in tuple; dirties buffer
- writes indication of deletion to transaction log

Vacuuming eventually compacts space in each page.

&lt;&lt; ∧ &gt;&gt;

# ❖ Updates in Heaps

SQL: **`update`** *R* **`set`** *F* = *val* **`where`** *Condition*

Analysis for updates is similar to that for deletion

- scan all pages
- replace any updated tuples (within each page)
- write affected pages to disk

$Cost_{update} = b_r + b_{qw}$

Complication: new tuple larger than old version (too big for page)

Solution: delete, re-organise free space, then insert

<< ^ >>

## ❖ Updates in Heaps (cont)

PostgreSQL tuple update:

```
heap_update(Relation relation,       // relation desc
            ItemPointer otid,        // old tupleID
            HeapTuple newtup, ...,   // new tuple data
            CommandId cid, ...)      // SQL statement
```

- essentially does **`delete(otid)`**, then **`insert(newtup)`**
- also, sets old tuple's **`ctid`** field to reference new tuple
- can also update-in-place if no referencing transactions

<< ∧ >>

## ❖ Heaps in PostgreSQL

PostgreSQL stores all table data in heap files (by default).

Typically there are also associated index files.

If a file is more useful in some other form:

- PostgreSQL may make a transformed copy during query execution
- programmer can set it via **`create index...using hash`**

Heap file implementation: src/backend/access/heap

<< ∧

## ❖ Heaps in PostgreSQL (cont)

PostgreSQL "heap file" may use multiple physical files

- files are named after the OID of the corresponding table
- first data file is called simply **`OID`**
- if size exceeds 1GB, create a fork called **`OID.1`**
- add more forks as data size grows (one fork for each 1GB)
- other files:
  - free space map (**`OID_fsm`**), visibility map (**`OID_vm`**)
  - optionally, TOAST file (if table has large varlen attributes)
- for details: Chapter 68 in PostgreSQL v12 documentation

Produced: 7 Mar 2021