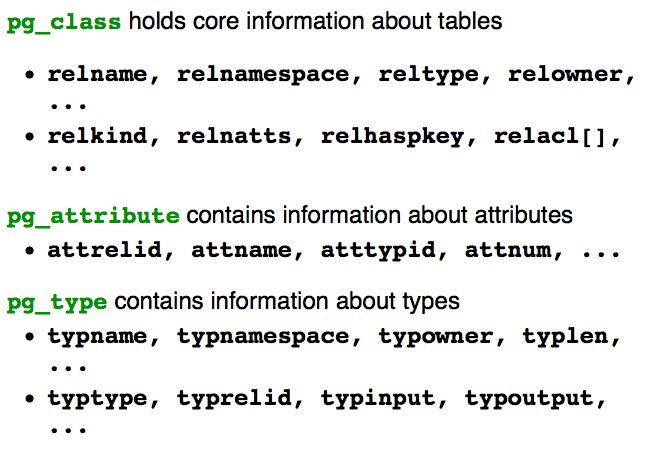
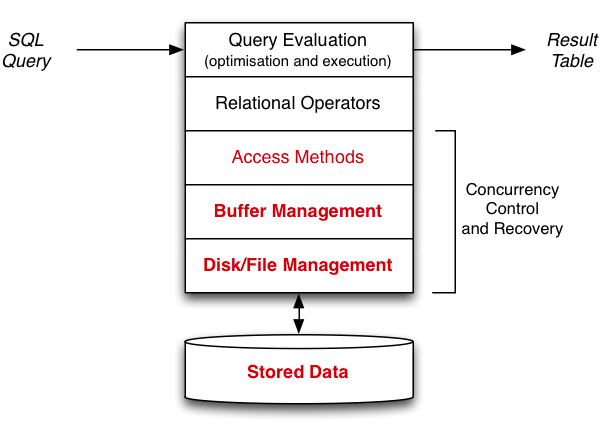
**Lecture 2**

**Catalogs** are tables describing database objects, e.g.



**DBMS Storage Manager**



**Storage Technology**

Persistent storage

* Large, cheap, relatively slow, accessed in blocks
* Used for long-term storage of data

Computational storage

* Small, expensive, fast, accessed by byte/word
* Used for all analysis of data

Access cost HDD:RAM ≈ 100000:1 (Hard Disk Drive: Random Access Memory)

* 100ms to read block containing two tuples
* 1μs to compare fields in two tuples

**Cost Models** (we compare costs of DB operations)

Important aspects in determining cost

* Data is always transferred to/from disk as whole blocks (pages)
* Cost of manipulating tuples in memory is negligible
* Overall cost determined primarily by #data-blocks read/written

Complicating factors in determining costs

* Not all page accesses require disk access (buffer pool)
* Tuples typically have variable size (tuples/page?)

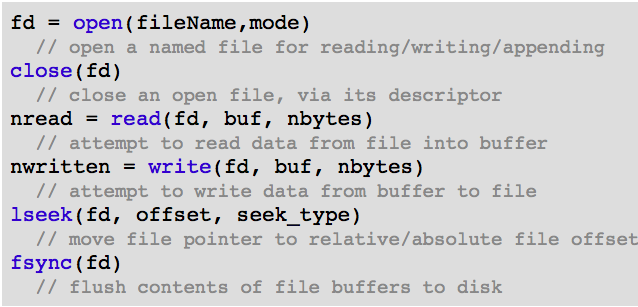
**File Management**

Aims of file management subsystem:

* organise layout of data within the filesystem
* handle mapping from database ID to file address
* transfer blocks of data between buffer pool and filesystem
* also attempts to handle file access error problems (retry)

Builds higher-level operations on top of OS file operations

Typical file operations provided by the operating system



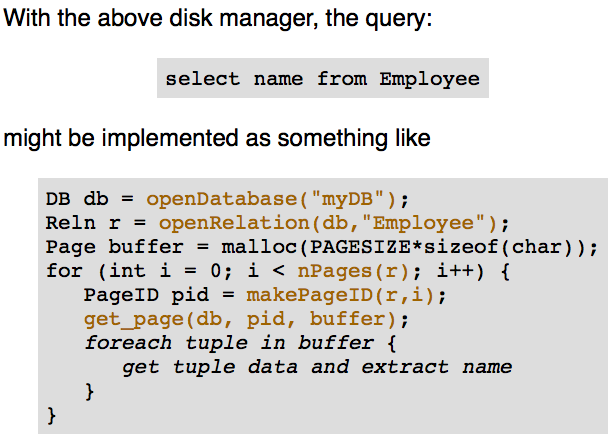
**Single-file DBMS**

Allocating space in Unix files is easy

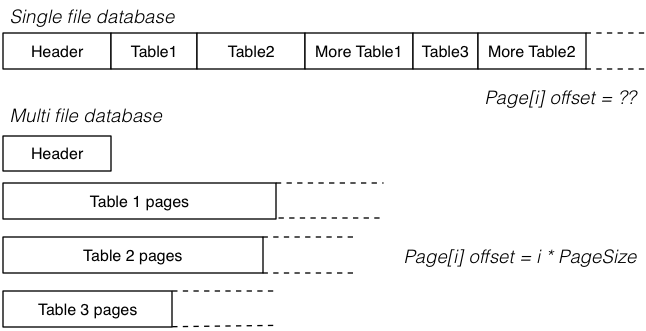
* Simply seek to the place you want and write the data
* If nothing there already, data is appended to the file
* If something there already, it gets overwritten

If the seek goes way beyond the end of the file

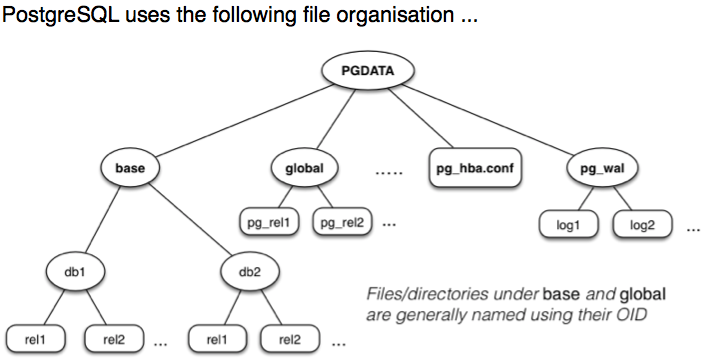
* Unix does not (yet) allocate disk space for the “hole”
* Allocates disk storage only when data is written there

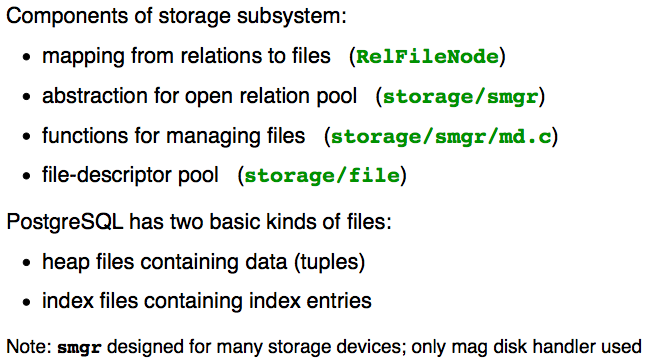


**Multi-file Disk Manager**

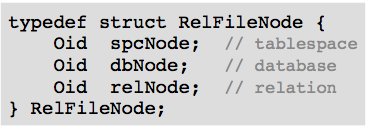


**PostgreSQL Storage Manager**



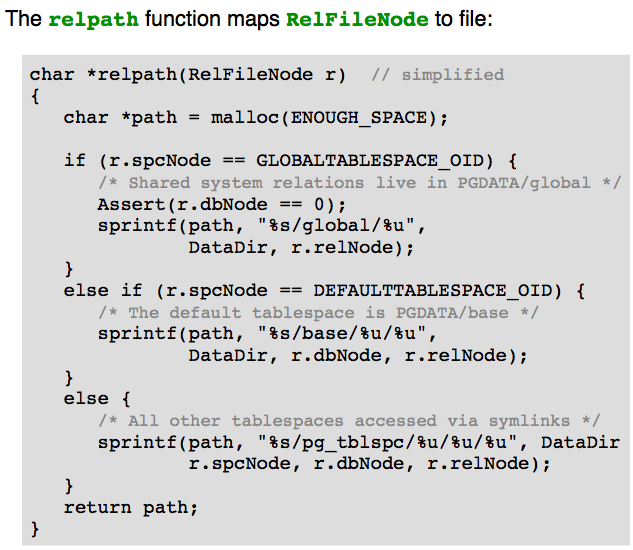


PostgreSQL identifies relation files via their OIDs



Global (shared) tables (e.g. pg\_database) have

* spcNode == GLOBALTABLESPACE\_OID
* dbNode == 0

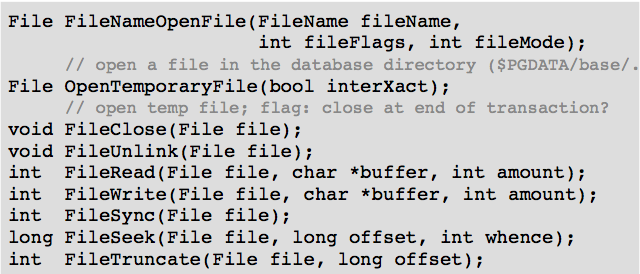


**File Descriptor Pool**

PostgreSQL maintains a pool of open file descriptors

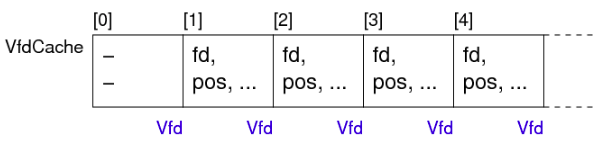
* File names – typedef char \*FileName
* Open files are referenced via – typedef int File

Interface to file descriptor (pool):

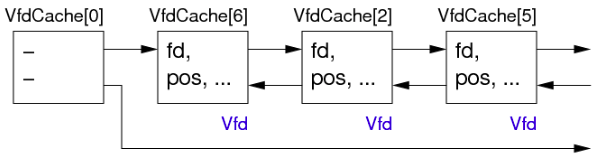


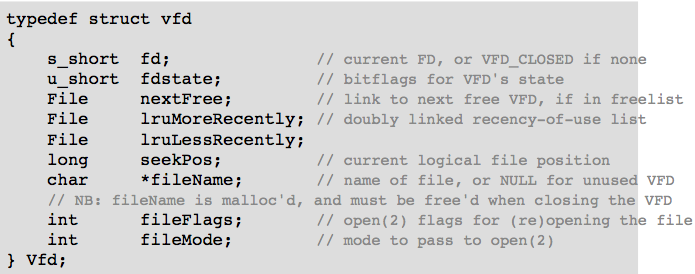
Virtual file descriptor (Vfd) (VfdCache[0] holds list head/tail pointer)

* Physically stored in dynamically-allocated array



* Also arranged into list by recency-of-use

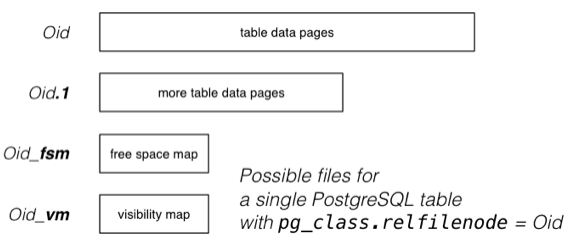




**File Manager**

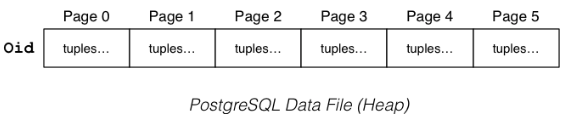
PostgreSQL stores each table

* In the directory PGDATA/pg\_database.oid
* Often in multiple files (aka forks)



Data files (Oid, Oid.1, …):

* Sequence of fixed-size blocks/pages (typically 8KB)
* Each page contains tuple data and admin data
* Max size of data files 1GB (Unix limitation)



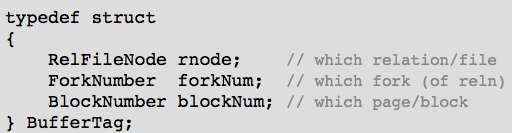
Free space map (Oid\_fsm):

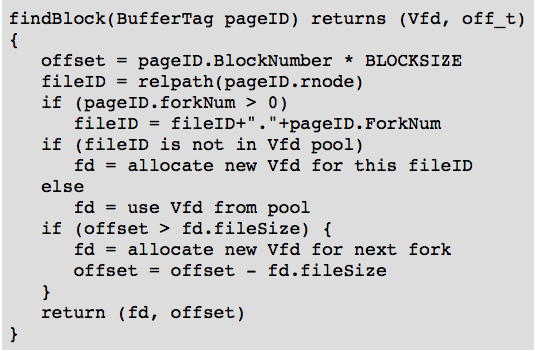
* Indicates where free space is in data pages
* “free” space is only free after VACUUM (DELETE simply marks tuples as no longer in use xmax)

Visibility map (Oid\_vm):

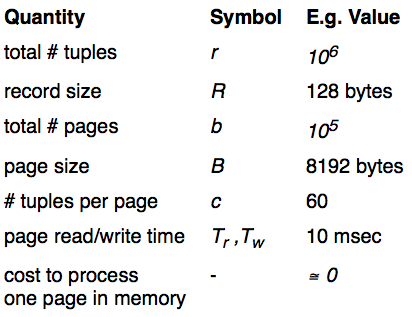
* Indicates pages where all tuples are “visible” (accessible to all currently active transactions)
* Such pages can be ignored by VACUUM
* Also used for index pages, to indicate all index entries visible

PostgreSQL **PageID** values are structured:





**DBMS Parameters**



**Buffer Pool**

Aim of buffer pool

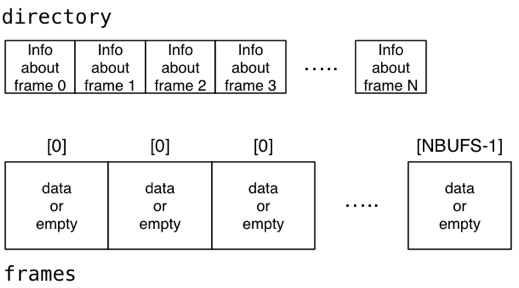
* hold pages read from database files, for possible re-use

Buffer pool operations

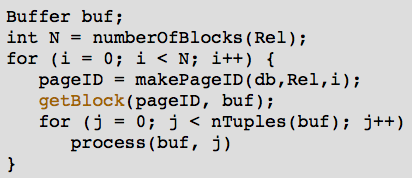
* request\_page(pid), release\_page(pid), …

Buffer pool data structures

* Page frames[NBUFS]; FrameData directory[NBUFS];
* Page is byte[BUFSIZE]; FrameData is struct {…}

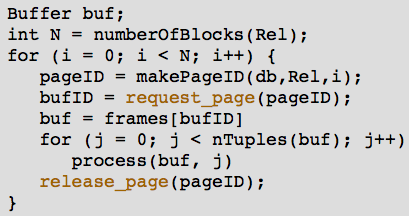


How scans are performed without Buffer Pool:



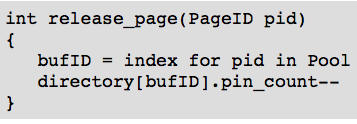
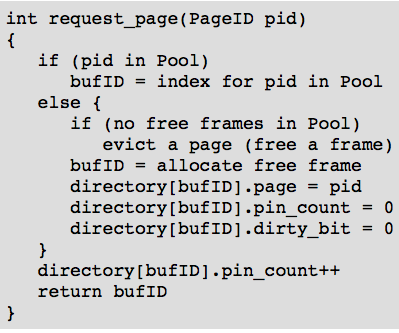
Requires N page reads, if we read it again, N page reads.

How scans are performed with Buffer Pool:



Requires N page reads on the first pass.

If we read it again, 0 ≤ page reads ≤ N



Evicting a page

* Find frame(s) preferably satisfying
  + Pin count = 0 (i.e. nobody using it)
  + Dirty bit = 0 (not modified)
* If selected frame was modified, flush frame to disk
* Flag directory entry as “frame empty”

if multiple frames can potentially be released

* Need a policy to decide which is best choice

**Page Replacement Policies**

* Least Recently Used (LRU)
* Most Recently Used (MRU)
* First In First Out (FIFO)
* Random