

Introduction to Postgres

Spiros Papadimitriou
spapadim@cs.cmu.edu

January 24, 2003

Overview

- ① **Installing Postgres**
- ② Running Postgres
- ③ Debugging Postgres
- ④ Buffer Manager
- ⑤ Miscellaneous

Source Distribution

- Unpack source tarball:

```
$ cd ~/src/  
$ tar -zxvf postgresql-7.2.2.tar.gz  
$ export PG_SRC=$HOME/src/postgresql-7.2.2
```

or, if running csh/tcsh:

```
$ setenv PG_SRC $HOME/src/postgresql-7.2.2
```

Source Distribution

- Run configure:

```
$ export PG_BIN=$HOME/pgsql  
$ ./configure --prefix=$PG_BIN --with-pgport=$PGPORT  
               --enable-depend  
               --enable-debug --enable-cassert  
               --with-maxbackends=8
```

- Fairly standard process (GNU automake/autoconf).
- Look in INSTALL for description of options (or `./configure --help`).

Source Distribution

- Compile and install:

```
$ make  
$ make install
```

- You may also want to

```
$ export PATH=$PG_BIN/bin:$PATH
```

Overview

- ① Installing Postgres
- ② **Running Postgres**
- ③ Debugging Postgres
- ④ Buffer Manager
- ⑤ Miscellaneous

Basic Initialization

- Environment variables

```
$ export PGPORT=nnnnn      # Pre-assigned
$ export PGDATA=$HOME/pgdata
```

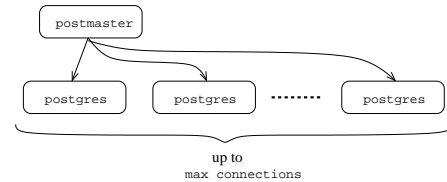
- Database cluster, etc. . .

```
$ initdb
```

```
$ pg_ctl start -l $PGDATA/serverlog -o "-i"
$ createdb test
$ pg_ctl stop
```

Postmaster

- Manages multiple client connections
 - Starts one backend process (postgres) per client.
- Backends communicate via shared memory.
 - Initialized by postmaster.



Bare Backend (Postgres)

```
iluvatar:~$ postgres test
DEBUG: database system was shut down at 2003-01-23 14:37:33 EST
DEBUG: checkpoint record is at 0/CF280C
DEBUG: redo record is at 0/CF280C;
      undo record is at 0/0; shutdown TRUE
DEBUG: next transaction id: 656; next oid: 186847
DEBUG: database system is ready

POSTGRES backend interactive interface
$Revision: 1.1.1.1.4.2 $ $Date: 2003/01/18 13:18:41 $

backend> select * from foo;

[...]
```

Basic Options

- Either as command line arguments to postgres. . .
- . . . or in \$PGDATA/postgresql.conf.
- Buffer pool size
 - pg_ctl start . . . -o "-i -B 16"
 - shared_buffers = 16
 - Must be at least 2×max_connections

Basic Options

- Optimizer—disable merge joins:
 - pg_ctl start . . . -o "-i -fm"
 - enable_mergejoin = false
- Optimizer—disable hash joins:
 - fh or enable_hashjoin = false

Overview

- ① Installing Postgres
- ② Running Postgres
- ③ **Debugging Postgres**
- ④ Buffer Manager
- ⑤ Miscellaneous

Log Messages

- `elog(DEBUG, ...)`
- Similar to `printf`, but flushes output to the logfile
- Examining log output:

```
$ tail -f $PGDATA/serverlog
ERROR: zero-length delimited identifier
DEBUG: pq_recvbuf: unexpected EOF on client connection
DEBUG: pq_recvbuf: unexpected EOF on client connection
DEBUG: smart shutdown request
DEBUG: shutting down
DEBUG: database system is shut down
```

Log Messages

- You may want to define a preprocessor conditional

```
#define MY_DEBUG_MSGS /* at top of file */
```

```
#ifdef MY_DEBUG_MSGS
elog(DEBUG, ...);
#endif
```

or possibly

```
#define MY_DEBUG_LEVEL 1 /* at top of file */
```

```
#if MY_DEBUG_LEVEL > 0
elog(DEBUG, ...);
#endif
```

Assert

- Catch bugs often, catch bugs early!
- `Assert(condition)`
- If condition is false, raises an (uncaught) `FailedAssertion` exception.

GDB / DDD

- Start backend in interactive mode from debugger
- DDD: graphical frontend for GDB
 - Graphical variable displays
 - Or use `print()` and `pprint()` (just for Nodes)


```
(gdb) set print elements 0 # no truncation
(gdb) call pprint(nodeptr)
```

BRIEF DEMO

Overview

- ① Installing Postgres
- ② Running Postgres
- ③ Debugging Postgres
- ④ **Buffer Manager**
- ⑤ Miscellaneous

Buffer Manager

- *Buffer pool*: A collection of pages in memory.
- *Page*: A fixed-size chunk of storage (default: 8Kb).
 - Fixed-size simplifies management—no external fragmentation.
 - Common approach (e.g., virtual memory vs. overlays).
- In other words: bufferpool is an *array* of pages.
- Also: a parallel array of metadata (“buffer descriptors”).
- “Co-operative caching”
 - No traps.
 - Explicit hints by bufferpool users.

Buffer Manager

- Pin
 - “Get me a page (if not there already)”
 - “I will be using it”
 - Pin count = counter of page “users”
- Unpin
 - “I’m done using the page”
 - “I (have / haven’t) modified it”
 - If pin count = 0, page can be evicted.
 - If modified, must first be written to disk.

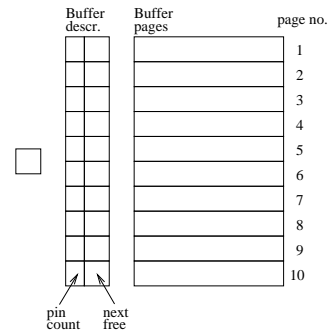
Replacement Policies

- When there are many unpinned pages, how do we select the one to evict?
- Random
- Least Recently Used (LRU)
- Most Recently Used (MRU)
- First-In First-Out (FIFO)
- Clock

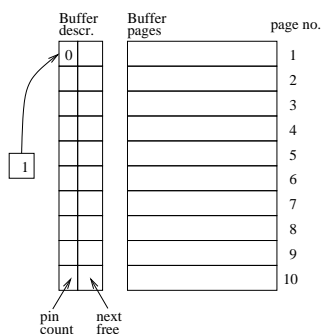
Replacement Policies—LRU

- Select least recently used page
- Used = ??
- Unpin = “I’m done using the page”
- When pin count becomes 0. . .
- Enqueue page
- Queue order = Last used order
- Victim = dequeue
- Queue implemented as a linked list (*freelist*)

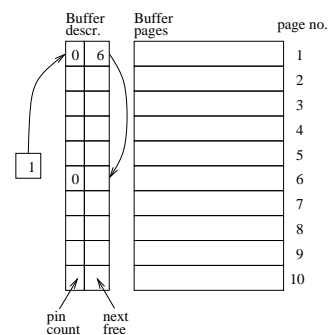
Replacement Policies—LRU



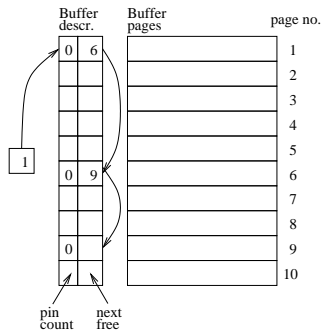
Replacement Policies—LRU



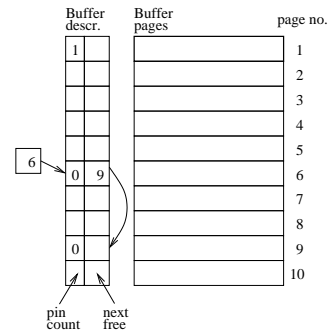
Replacement Policies—LRU



Replacement Policies—LRU



Replacement Policies—LRU



Replacement Policies—MRU

- *Most* recently used page
- What changes?
- Stack instead of queue
- Why?
 - Consider repeated sequential scan of $N + 1$ pages (N buffers). . .

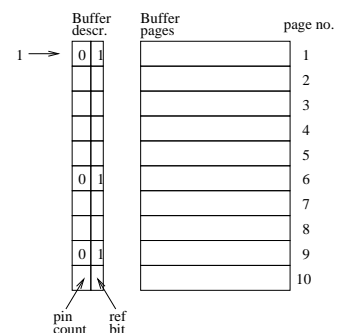
Replacement Policies—FIFO

- Least recently *brought in* page
- What changes?
- Queue again, but enqueue when pinning (not unpinning)
- Aside: Belady's anomaly (more buffers, more misses).

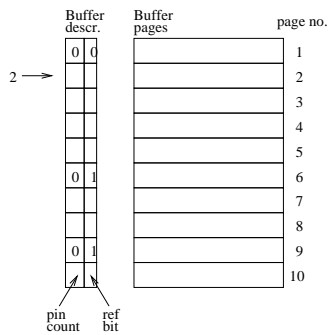
Replacement Policies—Clock

- Round-robin with a twist. . .
- Less space overhead
- Do not maintain complete order (freelist)
- Keep a "referenced" bit per page (set on last unpin)
- Visit *all* pages in bufferpool (round-robin):
 - If pinned, don't touch it
 - If unpinned and referenced, unset referenced bit
 - If unpinned and not referenced, select as victim
- Sometimes called "second chance"

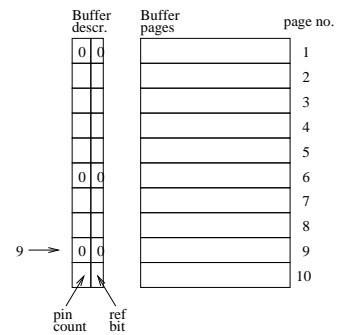
Replacement Policies—Clock



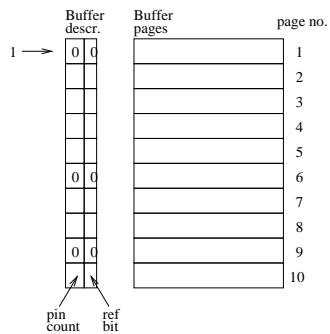
Replacement Policies—Clock



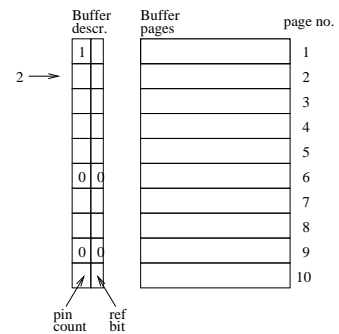
Replacement Policies—Clock



Replacement Policies—Clock



Replacement Policies—Clock



Replacement Policies—Clock

- Consider:
 - Instead of a flag (“referenced”)
 - A counter (“timestamp”)
 - * Incremented on every last unpin
 - * Decrementd
 - * Victim if “timestamp” = 0
- This is LRU; in Clock the timestamp is reduced to 1 bit.

Overview

- ① Installing Postgres
- ② Running Postgres
- ③ Debugging Postgres
- ④ Buffer Manager
- ⑤ **Miscellaneous**

C Trivia

Q. What does the following code do?

```
MASK_FREE = 2;
flags |= MASK_FREE;
```

A. Sets second bit of flags.

C Trivia

Q. How do you clear the second bit?

A.

```
flags &= ~MASK_FREE;
```

C Trivia

Q. What is wrong with the following code?

```
typedef struct node_t {
    int *data;
    struct node_t *next;
} node_t;
node_t *p;
...
/* p != NULL */
do {
    p = p->next;
    if (p->data == NULL)
        continue;
} while (*p->data < 5);
```

C Trivia

Q. What is wrong with the following code?

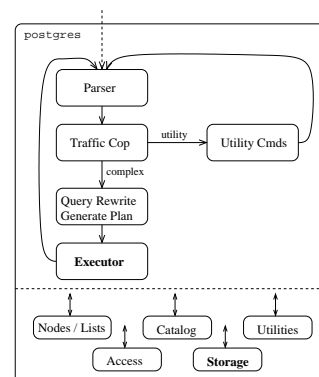
```
typedef struct node_t {
    int *data;
    struct node_t *next;
} node_t;
node_t *p;
...
/* p != NULL */
do {
    p = p->next;
    if (p == NULL)
        break;
    if (p->data == NULL)
        continue;
} while (*p->data < 5);
```

C Trivia

A. Careful about `continue` semantics: termination condition is evaluated. Use `goto` if you *absolutely* have to.

```
typedef struct node_t {
    int *data;
    struct node_t *next;
} node_t;
node_t *p;
...
/* p != NULL */
do {
    p = p->next;
    if (p == NULL)
        break;
} while (p->data == NULL || *p->data < 5);
```

Preview



Assignments:

1. Buffer manager (storage)
2. Query executor