Storing and Buffering Data

There is a new version of Problem Set #2. Download it, and copy your answers.

PS #2 submission is now open.

Boo and Kat's recitation tonight from 6pm-7pm.
Thoughts on WITH

The WITH SQL extension provides a means for referencing the same derived "table" multiple times in a SQL statement. You should use it as a temporary table:

**Good:**

```
WITH Reservations AS (  
    SELECT bid, COUNT(*) AS count  
    FROM Reserves  
    GROUP BY bid)
SELECT *  
FROM Reservations  
WHERE count=(SELECT MAX(count) FROM Reservations)
```

**Bad:**

```
WITH Reservations AS (  
    SELECT bid, COUNT(*) AS count  
    FROM Reserves  
    GROUP BY bid)
SELECT B.*, V.count  
FROM Boats B, Reservations V  
WHERE B.bid=V.bid
```
Thoughts on WITH

The WITH SQL extension a means for referencing the same derived "table" multiple times in a SQL statement. You should use it as a temporary table:

**Good:**

```sql
WITH Reservations AS (
    SELECT bid, COUNT(*) AS count
    FROM Reserves
    GROUP BY bid)
SELECT *
FROM Reservations
WHERE count=(SELECT MAX(count) FROM Reservations)
```

**Better:**

```sql
SELECT B.*, V.count
FROM Boats B,
     (SELECT bid, COUNT(*) AS count
      FROM Reserves
      GROUP BY bid) AS V
WHERE B.bid=V.bid
```

You shouldn't need "WITH" on PS #2.
Disks and Files

❖ A DBMS stores information in non-volatile storage.
  ▪ Magnetic Disks
  ▪ Solid State Disks
  ▪ Tapes
❖ This has major implications for DBMS design!
  ▪ **READ**: transfers from disk to main memory (RAM).
  ▪ **WRITE**: transfer from disk to RAM, change it, and then RAM to disk.
  ▪ Disk transfers are costly (slow) operations, relative to in-memory operations, so they must be planned and managed carefully!
Why Not Store Everything in Memory?

❖ **Costs too much.** $100 will buy you either 32GB of RAM or 4TB of disk today (125x).

❖ **Main memory is volatile.** We want data to be saved between runs. (Obviously!)

❖ **Data Size > Memory Size > Address Space**

❖ **Typical storage hierarchy:**
  - CPU Registers – temporary variables
  - Cache – Fast copies of frequently accessed memory locations (Cache and memory should indistinguishable)
  - Main memory (RAM) for currently used “addressable” data.
  - Disk for the main “big data” (secondary storage).
Storage Hierarchy

- **CPU Registers** – temporary program variables
- **Cache** – Fast copies of frequently accessed memory locations (Cache and memory are indistinguishable)
- **Main memory (RAM)** for currently “addressable” data.
- **Disk for files and databases** (secondary storage).
- **Tapes for archiving older versions of the data** (tertiary storage).
Disks

- Secondary storage device of choice.
- Main advantage over tapes: *random access* vs. *sequential*.
- Data is stored and retrieved in units called *disk blocks* or *pages*.
- Unlike RAM, time to retrieve a disk page can vary depending upon its location on disk.
  - Therefore, relative placement of pages on disk has major impact on DBMS performance!
Components of a Magnetic Disk

- The platters spin (say, 120rps).
- The arm assembly is moved in or out to position a head on a desired track. Tracks under heads make a cylinder (imaginary!).
- Only one head reads/writes at any one time.
- In the old days blocks corresponded to an angular region of the disk called a sector. These days there are more blocks along the outer tracks than the inner ones.
**Accessing a Disk Page**

- Time to access (read/write) a disk block:
  - *seek time* (moving arms to position disk head on track)
  - *rotational delay* (waiting for block to rotate under head)
  - *transfer time* (actually moving data to/from disk surface)

- Seek time and rotational delay dominate.
  - Seek time varies from about 2 to 15mS
  - Rotational delay from 0 to 8.3mS (ave 4.2mS)
  - Transfer rate is about 3.5mS per 256KB page (75 MB/sec)

- Key to lower I/O cost: *reduce seek/rotation delays!* Hardware vs. software solutions?
Arranging Pages on Disk

❖ Next block concept:
  - blocks on same track, followed by
  - blocks on same cylinder, followed by
  - blocks on adjacent cylinder
❖ Blocks in a file should be arranged sequentially on disk to minimize seek and rotational delays.
❖ For a sequential scan, pre-fetching several pages at a time is a big win!
Solid State Disk Drives

- A single transistor per 1-3 bits stored
- Data is read and written a page at a time, and erased a block at a time
- Typical block sizes:
  - 128 pages of 4,096+128 bytes each for a block size of 512 kB
- Timing:
  - Seek time: 0.08 to 0.16 mS
  - Rotational Delay: 0 mS
  - Transfer time: 0.5mS per 256Kb page (500 MB/S)
- ~$100 for 500 MB (8x more than a magnetic drive)
Disk Space Management

❖ Lowest layer of DBMS manages how space is used on disk. Abstraction unit is a “page”
❖ Higher levels call upon this layer to:
  ▪ allocate/de-allocate a page
  ▪ read/write a page
❖ Request for a *sequence* of pages must be satisfied by allocating the pages sequentially on disk! Higher levels don’t need to know how this is done, or how free space is managed.
❖ O/S Disk management vs. DBMS
**Buffer Management in a DBMS**

- Data must be in RAM for DBMS to operate on it!
- Table of `<frame#, pageid>` pairs is maintained. (i.e. which disk page is in which buffer pool frame)

---

**Page Requests from Higher Levels**

A Buffer Pool is just a Chunk of memory that holds "copies" of disk pages as needed by the DBMS. Usually thousands.

---

![Diagram of Buffer Pool and Memory Access]

- MAIN MEMORY
- DISK
- BUFFER POOL
- choice of frame dictated by replacement policy
When a Page is Requested ...

❖ If requested page is not in pool:
  ▪ Choose a frame for replacement
  ▪ If frame is dirty (its contents have been modified), write it to disk
  ▪ Read requested page into chosen frame

❖ Pin the page and return its address.

● If requests can be predicted (e.g., sequential scans) pages can be pre-fetched several pages at a time!
More on Buffer Management

❖ Requestor of page must *unpin* a frame when it is done, and indicate whether page has been modified:
  ▪ *dirty* bit is used for this.

❖ Some pages in the pool are be requested many times,
  ▪ Thus, a *pin count* is used. A page is a candidate for replacement iff *pin count* = 0.

❖ Crash recovery protocols may entail additional I/O when a frame is replaced.
  (*Write-Ahead Log* protocol; more later.)
Buffer Replacement Policy

❖ Frame is chosen for replacement by a replacement policy:
  ▪ Non-dirty, Least-recently-used LRU, FIFO, Clock, MRU etc.

❖ Policy can have big impact on # of I/O’s; depends on the access pattern.

❖ **Sequential flooding**: Nasty collision situation caused by LRU + repeated sequential scans.
  ▪ # buffer frames < # pages in file means each page request causes an I/O. MRU much better in this situation (but not in all situations, of course).
Sequential Flooding

Imagine $N$ frames are allocated for a table that occupies $N+1$ pages, and is accessed in an inner loop of a scan.
DBMS vs. OS File System

OS does disk space & buffer mgmt: why not let OS manage these tasks?

❖ Differences in OS support: portability issues
❖ Some limitations, e.g., files don’t span disks.
❖ Buffer management in DBMS requires ability to:
  ▪ pin a page in buffer pool, force a page to disk (important for implementing CC & recovery),
  ▪ adjust replacement policy, and pre-fetch pages based on access patterns in typical DB operations.
Record Formats: Fixed Length

How is data laid out within a block?

- Information about field types same for all records in a relation; stored in system catalogs.
- Finding $i^{th}$ field does not require scan of record.

![Diagram showing data layout within a block with fields F1, F2, F3, F4 and base address (B)].

Base address (B)  Address = B + L1 + L2
Record Formats: Variable Length

- Two alternative formats (# fields is fixed):

  Fields Delimited by Special Symbols

  Array of Field Offsets

  - Second offers direct access to i’th field, efficient storage of *nulls* (special *don’t know* value); small directory overhead.
Page Formats: Fixed Length Records

- **Record id** = `<page id, slot #>`. In first alternative, moving records for free space management changes `rid`; may not be acceptable.
Page Formats: Variable Length Records

- With a field directory you can reorder records without moving them. (key when building indices)
- You can also track "free space"
Files of Records

❖ Page or block is OK when doing I/O, but higher levels of DBMS operate on records, and files of records.

❖ **FILE**: A collection of pages, each containing a collection of records. Must support:
  - insert/delete/modify record
  - read a particular record (specified using record id)
  - scan all records (possibly with some conditions on the records to be retrieved)
Unordered (Heap) Files

- Simplest file structure contains records in no particular order.
- As file grows and shrinks, disk pages are allocated and de-allocated.
- To support record level operations, we must:
  - keep track of the *pages* in a file
  - keep track of *free space* on pages
  - keep track of the *records* on a page
- There are many alternatives for keeping track of this.
The header page ID and Heap file name must be stored somewhere.

Each page contains 2 ‘pointers’ plus data.
The entry for a page might also include the number of records and/or free bytes on the page.

The directory is itself a collection of pages; linked list implementation is just one alternative.

- Typically smaller than linked list of all HF pages!
System Catalogs

❖ For each relation:
  ▪ name, file name, file structure (e.g., Heap file)
  ▪ attribute name and type, for each attribute
  ▪ index name, for each index
  ▪ integrity constraints

❖ For each index:
  ▪ structure (e.g., B+ tree) and search key fields

❖ For each view:
  ▪ view name and definition

❖ Plus statistics, authorization, buffer pool size, etc.
  ● Catalogs are themselves stored as relations!
import sqlite3

db = sqlite3.connect("NFL.db")
cursor = db.cursor()
cursor.execute("SELECT * FROM sqlite_master")

for row in cursor:
    print([v for v in row])
SQLite_master

['table', 'Team', 'Team', 2, "CREATE TABLE Team(
   tid INTEGER PRIMARY KEY,
   mascot TEXT DEFAULT '')"]
['table', 'Player', 'Player', 3, 'CREATE TABLE Player(
    pid INTEGER PRIMARY KEY,
    name TEXT,
    height TEXT,
    weight INTEGER,
    college TEXT,
    dob DATE)'
]
['table', 'PlayedFor', 'PlayedFor', 4, 'CREATE TABLE PlayedFor(
    pid INTEGER,
    tid INTEGER,
    year INTEGER,
    position TEXT,
    jersey TEXT,
    games INTEGER,
    starts INTEGER,
    FOREIGN KEY(tid) REFERENCES Team(tid),
    FOREIGN KEY(pid) REFERENCES Player(pid),
    UNIQUE(pid,tid,year)']
['index', 'sqlite_autoindex_PlayedFor_1', 'PlayedFor', 5, None]
['table', 'TeamLocation', 'TeamLocation', 6, "CREATE TABLE TeamLocation(
    tid INTEGER,
    year INTEGER,
    place TEXT DEFAULT '',
    headcoach TEXT DEFAULT '',
    FOREIGN KEY(tid) REFERENCES Team(tid),
    UNIQUE(tid,year)"]
['index', 'sqlite_autoindex_TeamLocation_1', 'TeamLocation', 7, None]
['table', 'Draft', 'Draft', 8, 'CREATE TABLE Draft(
    pid INTEGER PRIMARY KEY,
    year INTEGER,
    round INTEGER,
    overall INTEGER,
    tid INTEGER,
    FOREIGN KEY(tid) REFERENCES Team(tid)
)'
]
['table', 'Game', 'Game', 1452, 'CREATE TABLE Game(
    season INTEGER,
    week TEXT,
    date DATE,
    vtid INTEGER,
    vscore INTEGER,
    htid INTEGER,
    hscore INTEGER,
    notes TEXT,
    FOREIGN KEY(vtid) REFERENCES Team(tid),
    FOREIGN KEY(htid) REFERENCES Team(tid),
    UNIQUE(season,week,htid)']
['index', 'sqlite_autoindex_Game_1', 'Game', 1453, None]
Summary

❖ Disks provide cheap, non-volatile storage.
  ▪ Random access, but cost depends on location of page on disk; important to arrange data sequentially to minimize seek and rotation delays.

❖ Buffer manager brings pages into RAM.
  ▪ Page stays in RAM until released by requestor.
  ▪ Written to disk when frame chosen for replacement (which is sometime after requestor releases the page).
  ▪ Choice of frame to replace based on replacement policy.
  ▪ Tries to pre-fetch several pages at a time.
**Summary (Contd.)**

- **DBMS vs. OS File Support**
  - DBMS needs features not found in many OS’s, e.g., forcing a page to disk, controlling the order of page writes to disk, files spanning disks, ability to control pre-fetching and page replacement policy based on predictable access patterns, etc.

- Variable length record format with field offset directory offers support for direct access to i’th field and null values.

- Slotted page format supports variable length records and allows records to move on page.
Summary (Contd.)

❖ File layer keeps track of pages in a file, and supports abstraction of a collection of records.
  ▪ Pages with free space identified using linked list or directory structure (similar to how pages in file are kept track of).

❖ Indexes support efficient retrieval of records based on the values in some fields.

❖ Catalog relations store information about relations, indexes and views. (Information that is common to all records in a given collection.)