Project Stuff

- Reminder: Phase II is due next week (Thursday, April 10)
- Read the project description and what each phase is asking for!
- After some discussion with the TA, I have simplified the project a bit and clarified some other parts of it, as follows:

- THINGS YOU NO LONGER NEED TO WORRY ABOUT:
  - the car dealership no longer leases cars, they only sell new cars
  - the car dealership no longer deals with contracts

Project Stuff (2)

- GENERAL ASSUMPTIONS:
  - you cannot assume away a requirement
  - credit checks and interactions with the DMV and bank (for getting a loan) are beyond the scope of this database system.
  - the dealership already has a stack of temporary tags, they do not need to contact the DMV for them. They will need to be able to print out a report of what temporary tag numbers were issued, and when they were issued, and insure that every temporary tag issued is different from every other one given out in the last 30 days.

Project Stuff (3)

- GENERAL ASSUMPTIONS continued:
  - the dealership may buy cars from more than one manufacturer, thus they may be selling more than one make of car. For example, Chevy dealerships also service and sell Geos; Dodge dealerships also service Plymouths, and so on. The number of different car manufacturers sold will be small (1, 2, 3 … not more than about 5 or so maximum)
  - the dealership only sells new cars, but it still deals with used cars when they are traded in. These cars are immediately shipped to a used car dealer which is conveniently located next door, so the new car dealership does not have to worry about keeping track of this used car on the lot.

Project Stuff (4)

- GENERAL ASSUMPTIONS continued:
  - Negotiations with banks for car loans are not handled by the dealership. Either a person buying a car will already come in with a loan from a bank, or some standard car loan methodology occurs that does not require any interaction with the database. From the perspective of the dealership, any car sale (regardless of loans) is like they are buying the car in full with cash.
  - the service department will work on cars that were not bought from the dealership, as long as it is a type they are able to service. For the project, you can assume any car brought for service will be a type of car they can work on.

Project Stuff (5)

- DATA ASSUMPTIONS (and hints):
  - all vehicles have VIN's (vehicle identification numbers) which are 17 alpha-numeric characters, so they will have to be input by a user.
  - customer identification numbers are NOT social security numbers.
  - links to other data are usually done by using key(s), so be careful for redundant data when a link via key(s) would be simpler. Data redundancy will bite you later on.

Project Stuff (6)

- DATA ASSUMPTIONS and hints (continued):
  - haggling is not part of the database, but a new car will have two prices associated with it: the manufacturers suggested retail price (MSRP) and the price the car was actually sold for. The cost of the car (how much the dealership paid the manufacturer for it) may be entirely different from both the above prices. Cars being repaired, but not bought from here will not need a price for the car

Project Stuff (7)

- TASK ASSUMPTIONS:
  - when ordering a vehicle from a manufacturer, you will not get immediate feedback. It is sort of like ordering from a paper catalog, you send in what you want (and your money) and eventually, you get what you ordered or a note saying it is out of stock. The actual transfer of money will be dealt with by the external billing system, but you must generate a record of the transaction when you make the order, and update it when the car comes in.
  - in order for a customer to by a custom car, either fixed up at the dealership or straight from the manufacturer or both, they have to buy the car first.

Project Stuff (8)

- TASK ASSUMPTIONS:
  - remember, there are two ways to customize a car: customize a car on the lot or order a custom car from the manufacturer. It is possible that customization will be done both ways for one car.

What Now?

- Most recently we worked on Chapter 11: Storage and File Structure.
- Now we start Chapter 12: Indexing and Hashing
  - 12.1 Basic Concepts
  - 12.2 Ordered Indices
  - 12.3 B+-tree Index Files
  - 12.4 B-tree Index Files
  - 12.5 Static Hashing
  - 12.6 Dynamic Hashing
  - 12.7 Comparison of Ordered Indexing and Hashing
  - 12.8 Index Definition in SQL
  - 12.9 Multiple-Key Access

Motivation

- Query response speed is a major issue in database design
- Some queries only need to access a very small proportion of the records in a database
  - "Find all accounts at the Perryridge bank branch"
  - "Find the balance of account number A-101"
  - "Find all accounts owned by Zephraim Cochrane"
- Checking every single record for the queries above is very inefficient and slow.
- To allow fast access for those sorts of queries, we create additional structures that we associate with files: ***indices*** (index files).

Basic Concepts

- Indexing methods are used to speed up access to desired data
  - e.g. Author card catalog in a library
- Search key -- an attribute or set of attributes used to look up records in a file. This use of the word ***key*** differs from that used before in class.
- An index file consists of records (index entries) of the form:

  (search-key, pointer)

  where the pointer is a link to a location in the original file
- Index files are typically much smaller than the original file
- Two basic types of index:
  - ***ordered indices***: search keys are stored in sorted order
  - ***hash indices***: search keys are distributed uniformly across "buckets" using a "hash function"

Index Evaluation Metrics

- We will look at a number of techniques for both ordered indexing and hashing. No one technique is best in all circumstances -- each has its advantages and disadvantages. The factors that can be used to evaluate different indices are:
- Access types supported efficiently.
  - Finding records with a specified attribute value
  - Finding records with attribute values within a specified range of values
- Access (retrieval) time. Finding a single desired tuple in the file.
- Insertion time
- Deletion time
- Update time
- Space overhead for the index

Index Evaluation Metrics (2)

- Speed issues are
  - Access time
  - Insertion time
  - Deletion time
  - Update time
- Access is the operation that occurs the most frequently, because it is also used for insert, delete, update
- For insert, delete, and update operations we must consider not only the time for the operation itself (inserting a new record into the file) but also any time required to update the index structure to reflect changes.
- We will often want to have more than one index for a file
  - e.g., card catalogs for author, subject, and title in a library

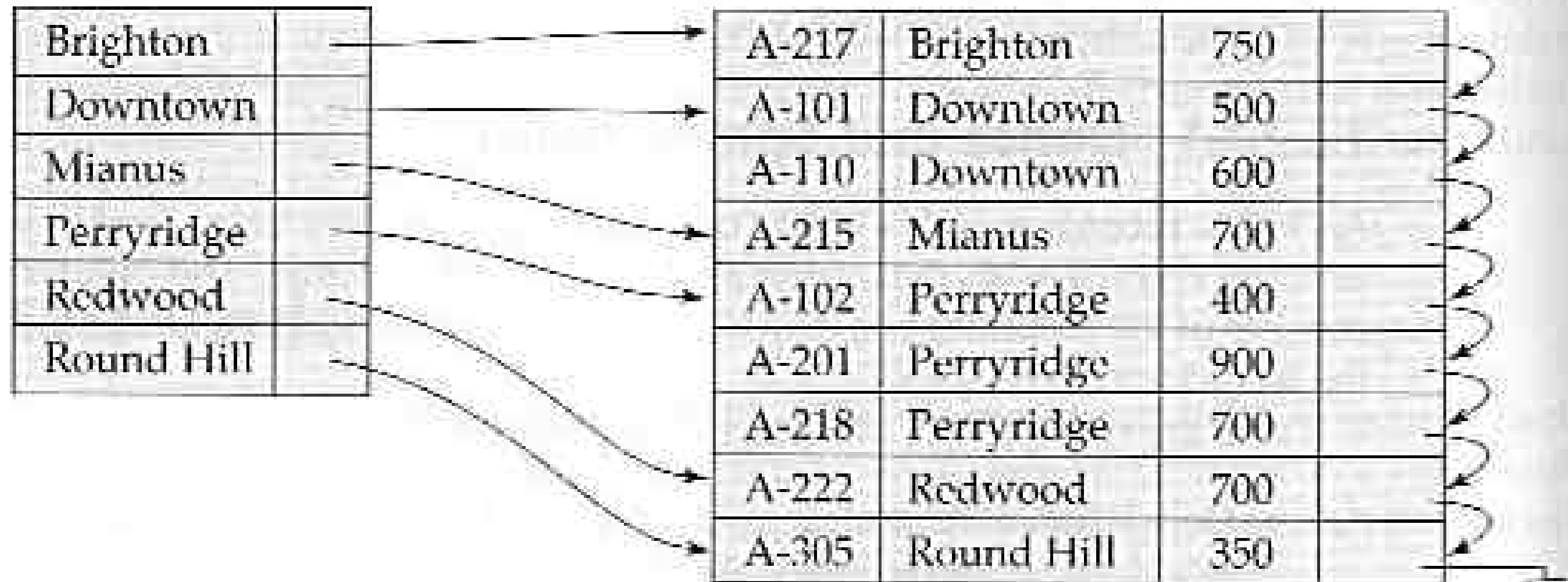
Ordered Indices

- An ordered index stores the values of the search keys in sorted order
- records in the original file may themselves be stored in some sorted order (or not)
- the original file may have several indices, on different search keys
- when there are multiple indices, a ***primary index*** is an index whose search key also determines the sort order of the original file. Primary indices are also called ***clustering indices***
- ***secondary indices*** are indices whose search key specifies an order different from the sequential order of the file. Also called non-clustering indices
- an ***index-sequential*** file is an ordered sequential file with a primary index.

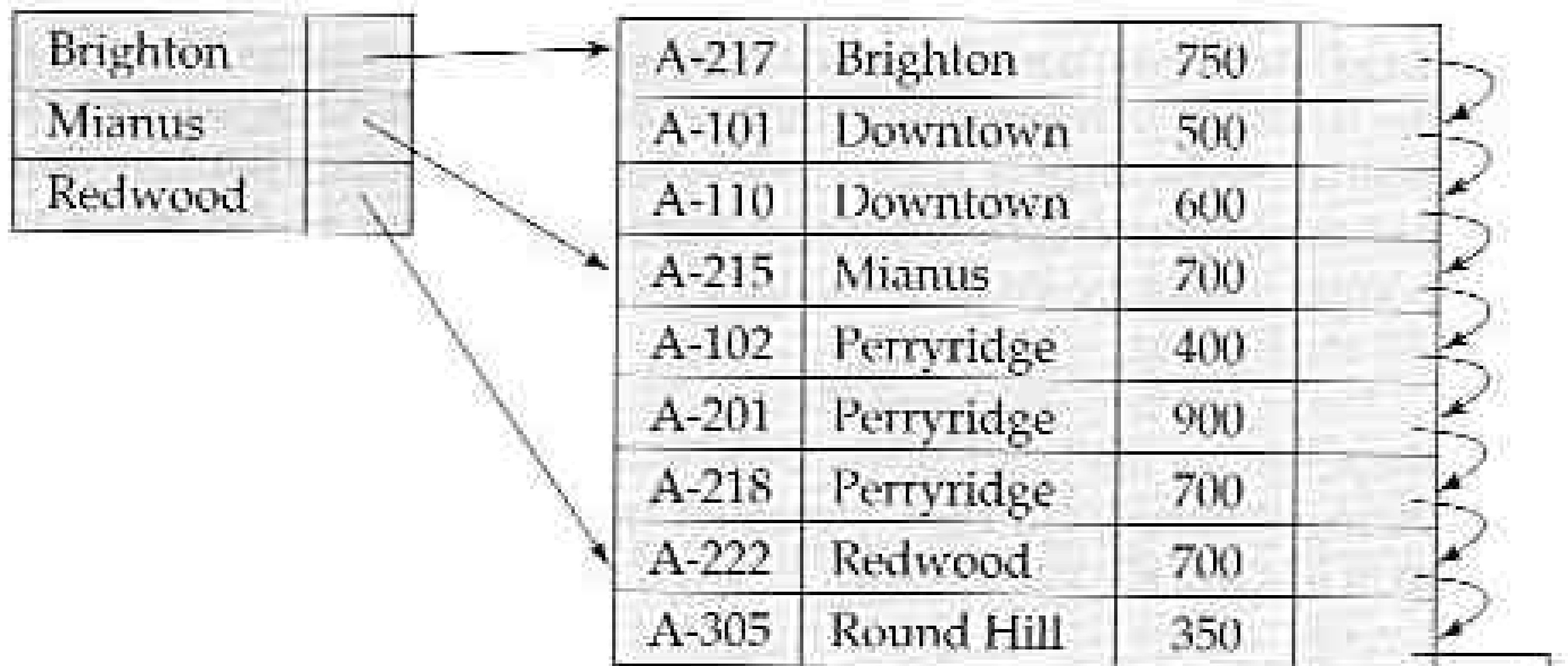
Dense and Sparse Indices

- A ***dense*** index is where an index record appears for every search-key value in the file
- A ***sparse*** index contains index records for only some search-key values
  - applicable when records are sequentially ordered on the search key
  - to locate a record with search-key value K we must:
    - find index record with largest search-key value <=K
    - search file sequentially starting at the location pointed to
    - stop (fail) when we hit a record with search-key value >K
  - less space and less maintenance overhead for insert, delete
  - generally slower than dense index for locating records

Example of a Dense Index



Example of a Sparse Index

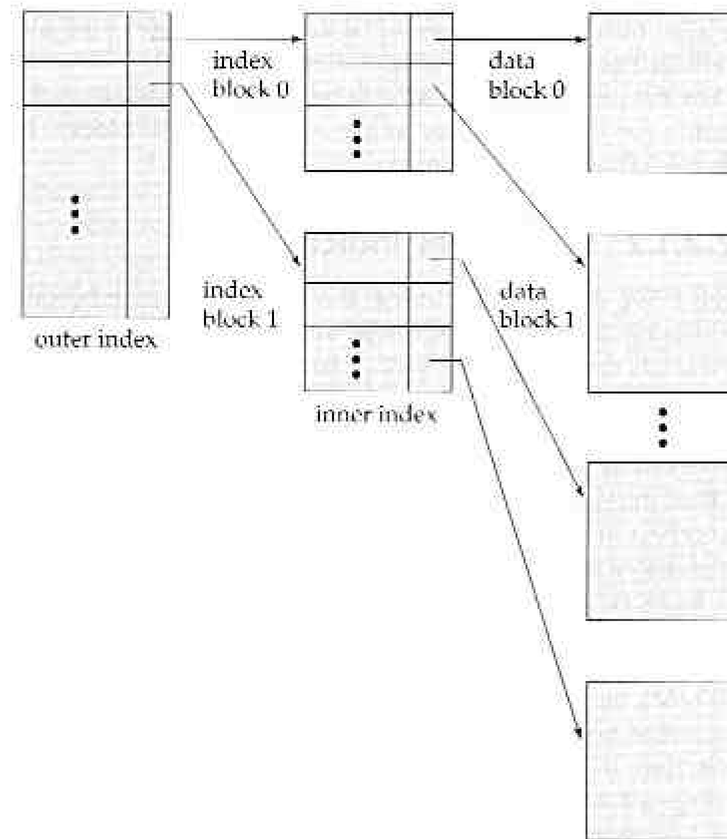


Problems with Index-Sequential Files

- Retrieve: search until the key value or a larger key value is found
  - individual key access BAD
  - scan the file in order of the key GOOD
- Insert is hard -- all higher key records must be shifted to place the new record
- Delete may leave holes
- Update is equivalent to a combined delete and insert
  - updating a search key field may cause the combined disadvantages of an insert and a delete by shifting the record's location in the sorted file
- differential files are often used to hold the recent updates until the database can be reorganized off-line

Multi-level Index

- If the primary index does not fit in memory, access becomes expensive (disk reads cost)
- To reduce the number of disk accesses to index records, we treat the primary index kept on disk as a sequential file and construct a sparse index on it
- If the outer index is still too large, we can create another level of index to index it, and so on
- Indices at all levels must be updated on insertion or deletion from the file



Index Update: Deletion

- When deletions occur in the primary file, the index will sometimes need to change
- If the deleted record was the only record in the file with its particular search-key value, the search-key is deleted from the index also
- Single-level index deletion:
  - dense indices -- deletion of search-key is similar to record deletion
  - sparse indices -- if an entry for the search key exists in the index, it is replaced by the next search key value in the original file (taken in search-key order)
  - Multi-level index deletion is a simple extension of the above

Index Update: Insertion

- As with deletions, when insertions occur in the primary file, the index will sometimes need to change
- Single-level index insertion:
  - first perform a lookup using the search-key value appearing in the record to be inserted
  - dense indices -- if the search-key value does not appear in the index, insert it
  - sparse indices -- depends upon the design of the sparse index. If the index is designed to store an entry for each block of the file, then no change is necessary to the index unless the insertion creates a new block (if the old block overflows). If that happens, the first search-key value in the new block is inserted in the index
  - multi-level insertion is a simple extension of the above

Secondary Index Motivation

- Good database design is to have an index to handle all common (frequently requested) queries.
- Queries based upon values of the primary key can use the primary index
- Queries based upon values of other attributes will require other (secondary) indices.

Secondary Indices vs. Primary Indices

- Secondary indices must be dense, with an index entry for every search-key value, and a pointer to every record in the file. A primary index may be dense or sparse. (*why*?)
- A secondary index on a candidate key looks just like a dense primary index, except that the records pointed to by successive values of the index are not sequential
- when a primary index is **not** on a candidate key it suffices if the index points to the first record with a particular value for the search key, as subsequent records can be found with a sequential scan from that point

Secondary Indices vs. Primary Indices (2)

- When the search key of a secondary index is not a candidate key (I.e., there may be more than one tuple in the relation with a given search-key value) it isn't enough to just point to the first record with that value -- other records with that value may be scattered throughout the file. A secondary index must contain pointers to all the records
  - so an index record points to a bucket that contains pointers to every record in the file with that particular search-key value

Secondary Index on *balance*

350
400
500
600
700
750
900

A-101	Downtown	500
A-217	Brighton	750
A-110	Downtown	600
A-215	Mianus	700
A-102	Perryridge	400
A-201	Perryridge	900
A-218	Perryridge	700
A-222	Redwood	700
A-305	Round Hill	350

Primary and Secondary Indices

- As mentioned earlier:
  - secondary indices must be dense
  - Indices offer substantial benefits when searching for records
  - When a file is modified, every index on the file must be updated
- this overhead imposes limits on the number of indices
  - relatively static files will reasonably permit more indices
  - relatively dynamic files make index maintenance quite expensive
- Sequential scan using primary index is efficient; sequential scan on secondary indices is very expensive
  - each record access may fetch a new block from disk (oy!)

Disadvantages of Index-Sequential Files

- The main disadvantage of the index-sequential file organization is that performance degrades as the file grows, both for index lookups and for sequential scans through the data. Although this degradation can be remedied through file reorganization, frequent reorgs are expensive and therefore undesirable.

- Next week we'll start examining index structures that maintain their efficiency despite insertion and deletion of data: the B+-tree (section 12.3)