Chapter 12: Part C

- Part A:
  - ★ Index Definition in SQL
  - ★ Ordered Indices
  - ★ Index Sequential
- Part B:
  - ★ B+-Tree Index Files
  - ★ B-Tree Index Files
- **Part C: Hashing**
  - ★ **Static and Dynamic Hashing**
  - ★ **Comparison of Ordered Indexing and Hashing**
  - ★ **Multi-Key Access**

Multiple-Key Access

- Use multiple indices for certain types of queries.
- Example:

  select *account-number*

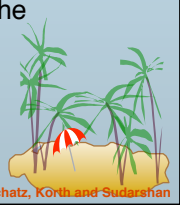
  from *account*

  where *branch-name* = “Perryridge” **and** *balance* - 1000
- Possible strategies for processing query using indices on single attributes:
  1. Use index on *branch-name* to find accounts with balances of $1000; test *branch-name* = “Perryridge”.
  2. Use index on *balance* to find accounts with balances of $1000; test *branch-name* = “Perryridge”.
  3. Use *branch-name* index to find pointers to all records pertaining to the Perryridge branch. Similarly use index on *balance*. Take intersection of both sets of pointers obtained.

Indices on Multiple Attributes

Suppose we have an index on combined search-key (*branch-name, balance*).

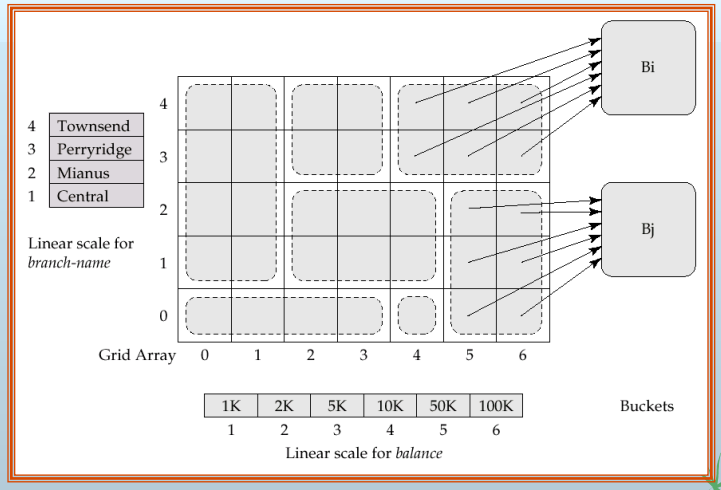
- With the **where** clause **where** *branch-name* = "Perryridge" **and** *balance* = 1000 the index on the combined search-key will fetch only records that satisfy both conditions.
  Using separate indices in less efficient — we may fetch many records (or pointers) that satisfy only one of the conditions.
- Can also efficiently handle **where** *branch-name* - "Perryridge" **and** *balance* < 1000
- But cannot efficiently handle **where** *branch-name* < "Perryridge" **and** *balance* = 1000
  May fetch many records that satisfy the first but not the second condition.



Grid Files

- Structure used to speed the processing of general multiple search-key queries involving one or more comparison operators.
- The grid file has a single grid array and one linear scale for each search-key attribute. The grid array has number of dimensions equal to number of search-key attributes.
- Multiple cells of grid array can point to same bucket
- To find the bucket for a search-key value, locate the row and column of its cell using the linear scales and follow pointer
- If a bucket becomes full, new bucket can be created if more than one cell points to it. If only one cell points to it, overflow bucket needs to be created.

Example Grid File for *account*



Grid Files (Cont.)

- A grid file on two attributes A and B can handle queries of the form $(a_1 \le A \le a_2)$, $(b_1 \le B \le b_2)$ as well as $(a_1 \le A \le a_2 \wedge b_1 \le B \le b_2)$, with reasonable efficiency.
- E.g., to answer $(a_1 \le A \le a_2 \wedge b_1 \le B \le b_2)$, use linear scales to find candidate grid array cells, and look up all the buckets pointed to from those cells.
- Linear scales must be chosen to uniformly distribute records across cells. Otherwise there will be too many overflow buckets.
- Periodic re-organization will help. But reorganization can be very expensive.
- Space overhead of grid array can be high.
- R-trees (Chapter 21) are an alternative

Partitioned Hashing

- Hash values are split into segments that depend on each attribute of the search-key.

  $(A_1, A_2, \ldots, A_n)$ for n attribute search-key

- Example: $n = 2$, for *customer,* search-key being (*customer-street, customer-city*)

search-key value	*hash value*
(Main, Harrison)	101 111
(Main, Brooklyn)	101 001
(Park, Palo Alto)	010 010
(Spring, Brooklyn)	001 001
(Alma, Palo Alto)	110 010

- To answer equality query on single attribute, need to look up multiple buckets. Similar in effect to grid files.

Other Multi Key Indices

- R* trees
- R+ trees (e.g., supported in ORACLE)
- Quad Trees
- Very important in spatio-temporal applications.