Outline

Goal: ‘Find similar / interesting things’
• Intro to DB
• Indexing - similarity search
• Data Mining

Indexing - Detailed outline
• primary key indexing
• secondary key / multi-key indexing
• spatial access methods
• text
• ...

Sec. key indexing
• attributes w/ duplicates (eg., EMPLOYEES, with ‘job-code’)
• Query types:
  – exact match
  – partial match
    • ‘job-code’= ‘PGM’ and ‘dept’=’R&D’
  – range queries
    • ‘job-code’=‘ADMIN’ and salary < 50K

Solution?

Sec. key indexing
• Query types - cont’d
  – boolean
    • ‘job-code’=’ADMIN’ or salary>20K
  – nn
    • salary ~ 30K
Solution?

- Inverted indices (usually, w/ B-trees)
- Q: how to handle duplicates?

```
salary-index
Name   Job-code Salary Dept
Smith  PGM   70   R&D
Jones  ADMIN 50  R&D
...    ...
Tomson ENG  50 SALES
```

Solution

- A#1: eg., with postings lists

```
salary-index
Name   Job-code Salary Dept
Smith  PGM   70   R&D
Jones  ADMIN 50 R&D
...    ...
Tomson ENG  50 SALES
```

• A#2: modify B-tree code, to handle dup’s

```
salary-index
Name   Job-code Salary Dept
Smith  PGM   70   R&D
Jones  ADMIN 50 R&D
...    ...
Tomson ENG  50 SALES
```

How to handle Boolean Queries?

• eg., ‘sal=50 AND job-code=PGM’?

```
salary-index
Name   Job-code Salary Dept
Smith  PGM   70   R&D
Jones  ADMIN 50 R&D
...    ...
Tomson ENG  50 SALES
```

– from indices, find lists of qual. record-ids
– merge lists (or check real records)

```
salary-index
Name   Job-code Salary Dept
Smith  PGM   70   R&D
Jones  ADMIN 50  R&D
...    ...
Tomson ENG  50 SALES
```

Sec. key indexing

• easily solved in commercial DBMS:

```
create index sal-index on EMPLOYEE (salary);
select * from EMPLOYEE
where salary > 50 and job-code = 'ADMIN'
```
**Sec. key indexing**

- can create combined indices:
  
  ```sql
  create index sj on EMPLOYEE( salary, job-code);
  ```

**Indexing - Detailed outline**

- primary key indexing
- secondary key / multi-key indexing
  - main memory: quad-trees
  - main memory: k-d-trees
- spatial access methods
- text
- ...

**Quad-trees**

- problem: find cities within 100mi from Pittsburgh
- assumption: all fit in main memory
- Q: how to answer such queries quickly?

**Quad-trees**

- A: recursive decomposition of space, e.g.:

  ```plaintext
  PGH
  •
  PHL
  •
  ATL
  ```
Quad-trees - search?

- find cities with \((35 < x < 45, 15 < y < 25)\):

  ![Diagram of Quad-trees - search]

Quad-trees - search?

- pseudocode:
  
  range-query( tree-ptr, range)
  if (tree-ptr == NULL) exit;
  if (tree-ptr->point within range){
    print tree-ptr->point
    for each quadrant {
      if ( range intersects quadrant ) {
        range-query( tree-ptr->quadrant-ptr, range);
      }
  }

Quad-trees - k-nn search?

- k-nearest neighbor algo - more complicated:
  - find ‘good’ neighbors and put them in a stack
  - go to the most promising quadrant, and update the stack of neighbors
  - until we hit the leaves

Quad-trees - discussion

- great for 2- and 3-d spaces
- several variations, like fixed decomposition:
  - ‘adaptive’
  - ‘fixed’

Quad-trees - discussion

- but: unsuitable for higher-d spaces (why?)
Quad-trees - discussion

- but: unsuitable for higher-d spaces (why?)
- A: 2^d pointers, per node!
- Q: how to solve this problem?
- A: k-d-trees!

Indexing - Detailed outline

- primary key indexing
- secondary key / multi-key indexing
  - main memory: quad-trees
  - main memory: k-d-trees
- spatial access methods
- text
  - ...
Indexing - Detailed outline

- primary key indexing
- secondary key / multi-key indexing
  - main memory: quad-trees
  - main memory: k-d-trees
    - insertion; deletion
    - range query; k-nn query
- spatial access methods
- text
- ...

k-d-trees - insertion

- Binary trees, with alternating ‘discriminators’

k-d-trees - deletion

- Tricky! ‘delete-and-promote’ (or ‘mark as deleted’)

k-d-trees - range query

- similar to quad-trees: check the root; proceed to appropriate child(ren).
**k-d-trees - k-nn query**

- e.g., 1-nn: closest city to ‘X’

![Diagram of k-d-trees - k-nn query 1](image1)

- A: check root; put in stack; proceed to child(ren)

![Diagram of k-d-trees - k-nn query 2](image2)

**Indexing - Detailed outline**

- primary key indexing
- secondary key / multi-key indexing
  - main memory: quad-trees
  - main memory: k-d-trees
  - insertion; deletion
  - range query; k-nn query
  - discussion
- spatial access methods
- text

![Diagram of Indexing - Detailed outline](image3)

**k-d trees - discussion**

- great for main memory & low ‘d’ (~<10)
- Q: what about high-d?
- A:
- Q: what about disk
- A:
Conclusions

- sec. keys: B-tree indices (+ postings lists)
- multi-key, main memory methods:
  - quad-trees
  - k-d-trees

References