ECS 165B: Database System Implementation
Lecture 11

UC Davis
April 21, 2010

Acknowledgements: portions based on slides by Raghu Ramakrishnan and Johannes Gehrke.

Class Agenda

- Last time:
  - Overview of DavisDB project, Part 2: indexing
- Today:
  - Query evaluation techniques: sorting
- Reading
  - Chapters 12 and 13 of Ramakrishnan and Gehrke (or Chapter 13 of Silberschatz et al)

Announcements

Code review sign-up sheet posted (see email I sent out for link); code reviews happening today through Monday

Repository updates: TestIX.cpp (sample tests for indexing); page file manager bugfixes; (not quite) final version of TestRM.cpp*

Grades for Part 1: Friday?

Discussion section Friday @11am**: B+ tree jam session**

Quiz #1 in class next Wednesday

Overview of Query Evaluation Techniques

Background material for Part 4 of the DavisDB project; some concepts we saw in Lecture 7 include:

- **evaluation plan** – relational algebra query drawn as a tree;
- **annotated evaluation plan** – each relational operator (e.g., "join") is annotated with the **physical operator** that will be used to perform the operation (e.g., "index nested loops join")
- **query optimizer** – takes a SQL query, produces an efficient annotated evaluation plan
- **query execution engine** – executes the annotated evaluation plan

Logical versus Physical Operators

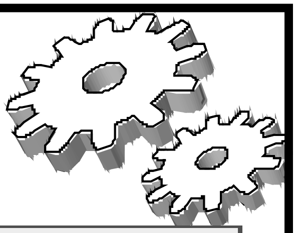
Logical operator	Physical operator
join: $E_1 \bowtie E_2$	nested loops join, index nested loops join, **sort-merge join**, …
projection: $\pi(E)$	projection
predicate: R	**file scan, index scan**, …
selection: $\sigma(E)$	selection
selection w/base predicate: $\sigma(R)$	**file scan with condition, index scan with condition**, …

- File scan (with condition): RecordFileScan (DavisDB Part 1)
- Index scan (with condition): IndexScan (Part 2)
  - Also underlies index nested loops join
- Others will be implemented in QueryEngine (Part 4)

Recall: Sort-Merge Join of *R* and *S*

- Sort R and S on the join column, then scan them to do a ``merge'' (on join col.), and output result tuples.
  - Advance scan of R until current R-tuple >= current S tuple, then advance scan of S until current S-tuple >= current R tuple; do this until current R tuple = current S tuple.
  - At this point, all R tuples with same value in Ri (*current R group*) and all S tuples with same value in Sj (*current S group*) *match*; output <r, s> for all pairs of such tuples.
  - Then resume scanning R and S.
- R is scanned once; each S group is scanned once per matching R tuple. (Multiple scans of an S group are likely to find needed pages in buffer.)

Example of Sort-Merge Join



sid	sname	rating	age
22	dustin	7	45.0
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

sid	bid	day	rname
28	103	12/4/96	guppy
28	103	11/3/96	yuppy
31	101	10/10/96	dustin
31	102	10/12/96	lubber
31	101	10/11/96	lubber
58	103	11/12/96	dustin

- Cost: M log M + N log N + (M+N)
  - The cost of scanning, M+N, could be M*N (very unlikely!)
- With 35, 100 or 300 buffer pages, both Reserves and Sailors can be sorted in 2 passes; total join cost: 7500.

Something to Consider in Part 2 (Indexing)

- In Part 4, **nested loops join** and **index nested loops join** will be the only join algorithms you will be required to implement
- Sort-merge join will be optional (XC), *but*, here's something to do in Part 2 that will make it easier
- Scan of B+ tree: **required** to return all record ids matching condition; **not required** to return them in order!
- May be a little extra work to have your scan return them in order, depending on details of your implementation…
- *But* this will let you use the index to do the sort for sort-merge join, if *R* and *S* are both indexed on the join attribute
  - We'll look at this again in a few slides

Plan for Upcoming Lectures

- Rest of today: we'll talk about **external sorting**, needed for sort-merge join, duplicate elimination, …

- Next lecture: we'll focus on the other physical query operators

- Subsequent lectures: generating physical plans (annotated evaluation plans) from logical plans (evaluation plans, aka relational algebra)