Goals for Today

- We gave a baseline for what join algorithms (and respective costs) were possible
- Use DB structures to expand optimization options

Recap – Plan Enumeration

- SQL Logical Plan
- Equ. Logical Plans
- Physical Plans

Recap – Disk Storage

- Can only read 1 block per read operation
  - Usually 512B to 4kB
- Sequential disk reads are faster than random ones
  - Cost ~1.2% random scan = full sequential scan

Recap – Making Cost Estimations

- RDBMS keeps statistics about our tables
  - B(R) = # of blocks in relation R
  - T(R) = # of tuples in relation R
  - V(attr, R) = # of distinct values of attr in R
Recap – Disk Storage

- Tables are stored as files
  - Heap file → Unsorted tuples
    - Nested Loop Joins
      - Block-at-a-time Nested Loop Join (cost = B(R)+B(R)*B(S))
      - Optimized Block-at-a-time Nested Loop Join (cost = B(R)+B(R)/N*B(S))
    - Hash Join B(R)<M (all of relation fits in memory)
      - cost = B(R)+B(S)
  - Sequential file → Sorted tuples

Outline

- Index structures
- Index join cost estimation
- Database tuning

Indexing

- Indexes (for this class) can be assumed to be already loaded into memory
- An index does not have to contain all tuple data
  - Only key values are stored in the index
  - If an index contains all tuple data it is called a "covering index"
- Indexes are access points for tables

Index Structures

- B+ Tree Index
  - Clustered
  - Unclustered
- Hash Index
- R Tree
- Radix Tree
- Bloom Filter
- Hilbert Curves
- …

What is a B Tree?

"What, if anything, the B stands for has never been established." – Wikipedia
- Search tree (like a binary search tree)
  - Nodes annotate max values
  - Large number of children per node
- Tree/node structure that is memory efficient
- Each entry of a node:
  - Left pointer to values less than entry
  - Right pointer to values greater than/equal to entry

What is a B Tree?
What is a B Tree?

Find the value 40

What is a B Tree?

Find the value 40

What is a B Tree?

Find the value 40

What is a B+ Tree Different?

• Leaf nodes point to data
  • Data is searchable by key value annotated by the node labels
• Leaf nodes form a linked list

Find the data associated with the key value 40
  (same search process)
Clusters vs Unclustered Index

- An index is either **clustered** or **unclustered**, depending on how the actual data is sorted on disk.

A **clustered index** is one that has the same key ordering as what is on disk (one per table).

- **An unclustered index** may exist without any ordering on disk (any number per table).

Benefits of B+ Trees

- **Range queries can be fast!**
  - Filtering a value on a valid range is essentially looking up some portion of a B+ tree.

Find the data associated with the key values 40 to 85.
Benefits of B+ Trees

• Range queries can be fast!
  - Filtering a value on a valid range is essentially looking up some portion of a B+ tree

Assuming uniform distribution of data values on numeric attribute a in table R, if the condition is:

- a = 4
- c1 < a < c2
- cond1 AND cond2

A selectivity factor will use a histogram.
Index-Based Selection

- For reference, a full sequential scan of data costs $B(R)$ I/Os (block reads)
- Provided some condition to read data:
  - Full sequential scan $\rightarrow B(R)$
  - Scan on clustered index $\rightarrow X \cdot B(R)$
  - Able to read a contiguous chunk of the file
  - Scan on unclustered index $\rightarrow X \cdot T(R)$
  - Worst case would read a different block everytime

Sequential Scan

Assume a block holds 4 tuples. I want tuples associated with values 40-85.
Without an index, finding a value must be done the “old fashioned way”

Disk

Assume a block holds 4 tuples. I want tuples associated with values 40-85.
Without an index, finding a value must be done the “old fashioned way”

Disk
Sequential Scan
Assume a block holds 4 tuples. I want tuples associated with values 40-85.
Without an index, finding a value must be done the “old fashioned way”

Total cost is \( B(R) \)

Unclustered Index Scan
Assume a block holds 4 tuples. I want tuples associated with values 40-85.
With an unclustered index, I start scanning tuples wherever they occur
Assume a block holds 4 tuples. I want tuples associated with values 40-85.
With an unclustered index, I start scanning tuples wherever they occur.

Sequential File with a different key or Heap File

Estimated cost is X * T(R)

Consider a query with X=1/10
Sequential scan = B(R) = 100
Index scan = X*T(R) = 1/10 * 10000 = 1000

Having indexes doesn’t mean you will see a speedup!
Sequential scan = 100
Index scan = 1000

Sequential disk reads are faster than random ones
Cost ~ 1-2% random scan = full sequential scan
Equivalent to selectivity ~.01-.02

Using an index in the wrong scenario can lead to a slowdown!
Common example: Full sequential scan vs unclustered index scan with high X value and/or small tuple size (large T(R):B(R) ratio)

CREATE TABLE Users (id INT, age INT, score INT);
CREATE INDEX U_age ON Users(age);
CREATE INDEX U_age_score ON Users(age, score);
CREATE CLUSTERED INDEX U_score_age ON Users(score, age);
Create Indexes in SQL

CREATE TABLE Users (id INT, age INT, score INT);

CREATE INDEX U_age ON Users(age)

CREATE INDEX U_age_score ON Users(age, score)

CREATE CLUSTERED INDEX U_score_age ON Users(score, age)

Unclustered by default

Order specifies precedence in sorting

Unclustered by default

Order specifies precedence in sorting

Unclustered by default

Order specifies precedence in sorting

Can’t scan per block of R since tuples in blocks don’t have the same attribute values

Can’t scan per block of R since tuples in blocks don’t have the same attribute values

Can’t scan per block of R since tuples in blocks don’t have the same attribute values

Can’t scan per block of R since tuples in blocks don’t have the same attribute values

Index-Based Equijoin

- Assume index exists on the join attribute a of S

| SELECT * FROM R, S WHERE R.a = S.a |
| (?? Index Join) |

- Clustered Index Join
  - Perform a clustered index scan for each tuple of R
  - B(R)+T(R)(X*B(S)) = B(R)+T(R)(B(S)/V(a, S))

- Unclustered Index Join
  - Perform an unclustered index scan for each tuple of R
  - B(R)+T(R)(X*T(S)) = B(R)+T(R)(T(S)/V(a, S))
Leveraging Indexes

- Often for applications, workloads can be well described
  - Flights application
  - Search method → query on city name values
  - Data visualization software (e.g. Tableau)
  - 2D plot → query on graph axis bounds
- Create indexes to match expected query workload

CREATE TABLE Users (
id INT PRIMARY KEY,
age INT,
score INT,
...
);

What indexes could we make on Users?

Things to consider:
- What is the expected result size for each query?
- Do either of these queries need to be returned ASAP?

IDs are unique so an unclustered index would do fine.

Without more information, default to clustering on the index that will be used more (clustered index on score)

CREATE TABLE Users (
id INT PRIMARY KEY,
age INT,
score INT,
...
);

What indexes could we make on Users?

CREATE TABLE Users (
id INT PRIMARY KEY,
age INT,
score INT,
...
);

What indexes could we make on Users?

CREATE TABLE Users (
id INT PRIMARY KEY,
age INT,
score INT,
...
);

What indexes could we make on Users?
Leveraging Indexes

CREATE TABLE Users (id INT PRIMARY KEY, age INT, score INT, ...);

What indexes could we make on Users?

Hack:  
• Create a covering index primarily keyed on score
• Create a covering index primarily keyed on age

IDs are unique so an undistributed index would do fine.

Essentially a sorted copy of the table. Fast but space inefficient and table updates are slow.

Database design question

• Choosing how to configure a database system is an interesting (i.e. hard) problem
• A database that is used by many people will often need one or more dedicated personnel to manage it (Database Administrator)  
  • Logical design (multi-team coordination)
  • Physical design (hardware and system considerations)
  • Permission management (visibility and security)
  • Integration (company acquisitions and mergers)
  • ...

Multiple Joins

• Pipelined Execution
  • Tuples are processed through the entire query plan
  • Fast

Pipelined Execution

• Iterator interface of RA operators (Volcano Iterator Model)
  • open() on every operator at start
  • close() on every operator at end
  • next() to get the next tuple from a child operator or input table

Pipelined Execution Example
Pipelined Execution Example

- Iterator interface of RA operators (Volcano Iterator Model)

```
∪.⋈<.σB.<C.open()
```
Pipelined Execution Example

- Iterator interface of RA operators (Volcano Iterator Model)

Next implementation will depend on algorithm used (CSE 444)
Pipelined Execution Example

- Iterator interface of RA operators (Volcano Iterator Model)

Volcano!!!