Introduction to Data Management

Database Tuning

Alyssa Pittman
Based on slides by Jonathan Leang, Dan Suciu, et al
Paul G. Allen School of Computer Science and Engineering
University of Washington, Seattle

November 1, 2019
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
```
SELECT * FROM T, R, S
WHERE ...
```

Logical Plan

Eq. Logical Plans

RDBMS

Least Cost Plan

Execution

Physical Plans

Indexes

November 1, 2019
Recap – Assumptions

For this class we make a lot of assumptions

▪ **Disk-based storage**
  - HDD not SDD

▪ **Row-based storage**
  - Tuples are stored contiguously

▪ **IO cost** only
  - One disk access is ~100000x more expensive than one main memory access

▪ **Cold cache**
  - No data preloaded into main memory
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
  - \( B(R) = \# \text{ of blocks} \) in relation \( R \)
  - \( T(R) = \# \text{ of tuples} \) in relation \( R \)
  - \( V(\text{attr}, R) = \# \text{ of distinct values} \) for \( \text{attr} \) 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))
      - Block-Nested-Loop Join (cost = B(R)+B(R)/N*B(S))
    - Hash Join (B(R)<M, cost = B(R)+B(S))
    - Sort-Merge Join (B(R)+B(S)<M, cost = B(R)+B(S))
  - **Sequential file** → Sorted tuples
Outline

- Index structures
- Index join cost estimation
- Database tuning
Indexing

An **index** is an additional file allowing **fast access** to records given a **search key**.

It stores **(key, value)** pairs:

**(attribute, pointer to the record)**

<table>
<thead>
<tr>
<th>Key</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>John</td>
<td>Doe</td>
</tr>
<tr>
<td>20</td>
<td>Jane</td>
<td>Doe</td>
</tr>
<tr>
<td>50</td>
<td>Alice</td>
<td>Foo</td>
</tr>
<tr>
<td>200</td>
<td>Bob</td>
<td>Bar</td>
</tr>
</tbody>
</table>
Which key?

- Primary key: uniquely identifies a tuple
- Candidate Keys: other keys defined on relations
- Key of the sequential file: how the data file is sorted, if at all
- Index key: how the index is organized
Example: Student, sorted data file

### Index student_id on Student.id

<table>
<thead>
<tr>
<th>ID</th>
<th>FirstName</th>
<th>LastName</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>John</td>
<td>Doe</td>
</tr>
<tr>
<td>20</td>
<td>Jane</td>
<td>Doe</td>
</tr>
<tr>
<td>50</td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td></td>
<td></td>
</tr>
<tr>
<td>230</td>
<td></td>
<td></td>
</tr>
<tr>
<td>400</td>
<td></td>
<td></td>
</tr>
<tr>
<td>410</td>
<td></td>
<td></td>
</tr>
<tr>
<td>412</td>
<td></td>
<td></td>
</tr>
<tr>
<td>500</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Logical relation

<table>
<thead>
<tr>
<th>ID</th>
<th>FirstName</th>
<th>LastName</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>John</td>
<td>Doe</td>
</tr>
<tr>
<td>20</td>
<td>Jane</td>
<td>Doe</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Example: Student, unsorted data file

Index student_id on Student.id

<table>
<thead>
<tr>
<th>ID</th>
<th>FirstName</th>
<th>LastName</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>John</td>
<td>Doe</td>
</tr>
<tr>
<td>20</td>
<td>Jane</td>
<td>Doe</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Logical relation

Data file student

<table>
<thead>
<tr>
<th>ID</th>
<th>FirstName</th>
<th>LastName</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>John</td>
<td>Doe</td>
</tr>
<tr>
<td>20</td>
<td>Jane</td>
<td>Doe</td>
</tr>
<tr>
<td>230</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
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”
- A table can have multiple indexes
Index Structures

▪ **Hash Index**
  ▪ B+ Tree Index
    • Clustered
    • Unclustered

▪ R Tree

▪ Radix Tree

▪ Bloom Filter

▪ Hilbert Curves

▪ Inverted Index

▪ ...
Hash Index

Index student_id on Student.id

Data file student

Logical relation

<table>
<thead>
<tr>
<th>ID</th>
<th>FirstName</th>
<th>LastName</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>John</td>
<td>Doe</td>
</tr>
<tr>
<td>20</td>
<td>Jane</td>
<td>Doe</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Index Structures

- Hash Index
- **B+ Tree Index**
  - Clustered
  - Unclustered
- R Tree
- Radix Tree
- Bloom Filter
- Hilbert Curves
- Inverted Index
- ...
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
What is a B Tree?

- A B Tree is a self-balancing tree data structure that maintains sorted keys and allows searches, insertions, and deletions in logarithmic time.

- Each node in a B Tree can have at most two children and at most two keys.

- The B Tree is balanced, meaning all leaf nodes are at the same level or one level apart.

- The following diagram illustrates a 4-way B Tree with keys 10, 15, 18, 20, 30, 40, 50, 60, 65, 80, 85, 90, and 100.
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?

Find the value 40
How 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
How is a B+ Tree Different?

Find the data associated with the key value 40 (same search process)
• An **unclustered index** may exist without any ordering on disk (any number per table)

**Sequential File with a different key** or **Heap File**
A **clustered index** is one that has the **same key ordering** as what is on disk (one 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

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

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

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

Find the data associated with the key values 40 to 85

Diagram showing data structure and key values

10 15 18
20 30 40 50
60 65
80 85 90
100 120 140

d1 d2 d3 d4 d5 d6 d7 d8 d9 d10 d11 d12 d13
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
Estimating Amount of Data Read

- **Selectivity Factor** \((X)\) \(\rightarrow\) Proportion of total data needed

- Assuming uniform distribution of data values on numeric attribute \(a\) in table \(R\), if the condition is:
  - \(a=c\) \(\rightarrow\) \(X \approx \frac{1}{V(a,R)}\)
  - \(a<c\) \(\rightarrow\) \(X \approx \frac{c-\min(a,R)}{\max(a,R)-\min(a,R)}\)
  - \(c_1<a<c_2\) \(\rightarrow\) \(X \approx \frac{c_2-c_1}{\max(a,R)-\min(a,R)}\)
  - \(\text{cond1 AND cond2}\) \(\rightarrow\) \(X \approx X_1 \times X_2\)

- Disclaimer: More thorough selectivity estimation will use a histogram
Estimating Amount of Data Read

- **Selectivity Factor** \( X \rightarrow \) Proportion of total data needed

- Assuming uniform distribution of data values on numeric attribute \( a \) in table \( R \), if the condition is:
  - \( a = c \rightarrow X \approx \frac{1}{V(a,R)} \)
  - \( a < c \rightarrow X \approx \frac{c - \min(a,R)}{\max(a,R) - \min(a,R)} \)
  - \( c_1 < a < c_2 \rightarrow X \approx \frac{c_2 - c_1}{\max(a,R) - \min(a,R)} \)
  - cond1 AND cond2 \( X \approx X_1 \times X_2 \)

- Disclaimer: More thorough selectivity estimation will use a histogram
Estimating Amount of Data Read

- **Selectivity Factor** \( X \) \( \rightarrow \) Proportion of total data needed

- Assuming uniform distribution of data values on numeric attribute \( a \) in table \( R \), if the condition is:
  - \( a = c \) \( \rightarrow \) \( X \cong \frac{1}{V(a,R)} \) \( a < 4 \)
  - \( a < c \) \( \rightarrow \) \( X \cong \frac{c - \text{min}(a,R)}{\text{max}(a,R) - \text{min}(a,R)} \) \( [1, 1, 2, 2, 3, 3, 4, 4, 5, 5] \)
  - \( c_1 < a < c_2 \) \( \rightarrow \) \( X \cong \frac{c_2 - c_1}{\text{max}(a,R) - \text{min}(a,R)} \) \( X = \frac{4 - 1}{5 - 1} \)
  - \( \text{cond1 AND cond2} \) \( \rightarrow \) \( X \cong X_1 \ast X_2 \)

- Disclaimer: More thorough selectivity estimation will use a histogram
Estimating Amount of Data Read

- **Selectivity Factor** \( X \) \( \rightarrow \) Proportion of total data needed

- Assuming uniform distribution of data values on numeric attribute \( a \) in table \( R \), if the condition is:
  - \( a = c \) \( \rightarrow \) \( X \approx \frac{1}{V(a,R)} \) \ 2 < a < 4
  - \( a < c \) \( \rightarrow \) \( X \approx \frac{c - \min(a,R)}{\max(a,R) - \min(a,R)} \) \[1, 1, 2, 2, 3, 3, 4, 4, 5, 5\]
  - \( c_1 < a < c_2 \) \( \rightarrow \) \( X \approx \frac{c_2 - c_1}{\max(a,R) - \min(a,R)} \) \( X = \frac{4 - 2}{5 - 1} \)
  - \( \text{cond1 AND cond2} \) \( \rightarrow \) \( X \approx X_1 \times X_2 \)

- Disclaimer: More thorough selectivity estimation will use a histogram
Index-Based Selection

- For reference, a full sequential scan of data costs $B(R)$ IOs

- Provided some condition to read data:
  - Full **sequential scan** $\rightarrow B(R)$
  - Scan on **clustered index** $\rightarrow X^*B(R)$
    - Able to read a contiguous chunk of the file
  - Scan on **unclustered index** $\rightarrow X^*T(R)$
    - 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”
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”
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”
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”
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”
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: $B(R)$
Assume a block holds 4 tuples. I want tuples associated with values 40-85. With a clustered index, I start scanning blocks in the range they are at.
Assume a block holds 4 tuples. I want tuples associated with values 40-85. With a clustered index, I start scanning blocks in the range they are at.
Clustered Index Scan

Assume a block holds 4 tuples. I want tuples associated with values 40-85. With a clustered index, I start scanning blocks in the range they are at.

Total cost: $X \cdot B(R)$
Unclustered Index Scan

Assume a block holds 4 tuples. I want tuples associated with values 40-85. With an unclustered index, I scan tuples wherever they occur.

Sequential File with a different key or Heap File
Unclustered Index Scan

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

Assume a block holds 4 tuples. I want tuples associated with values 40–85. With an unclustered index, I scan tuples wherever they occur.

Sequential File with a different key or Heap File
Unclustered Index Scan

Assume a block holds 4 tuples. I want tuples associated with values 40-85. With an unclustered index, I scan tuples wherever they occur.

Sequential File with a different key or Heap File
Unclustered Index Scan

Assume a block holds 4 tuples. I want tuples associated with values 40-85. With an unclustered index, I scan tuples wherever they occur.

Sequential File with a different key or Heap File
Unclustered Index Scan

Assume a block holds 4 tuples. I want tuples associated with values 40-85. With an unclustered index, I scan tuples wherever they occur.

Total cost: $X \times T(R)$

Sequential File with a different key or Heap File
Index Expectations

- 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)

  Known:
  
  $B(R) = 100$
  
  $T(R) = 10000$

  Consider a query with $X=1/10$

  **Sequential scan**
  
  $= B(R)$
  
  $= 100$

  **Index scan**
  
  $= X * T(R)$
  
  $= 1/10 * 10000$
  
  $= 1000$
Index Expectations

- 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)

Having indexes doesn’t mean you will see a speedup!

Known:

- $B(R) = 100$
- $T(R) = 10000$

Consider a query with $X=1/10$

<table>
<thead>
<tr>
<th>Sequential scan</th>
<th>Index scan</th>
</tr>
</thead>
<tbody>
<tr>
<td>$= B(R)$</td>
<td>$= X*T(R)$</td>
</tr>
<tr>
<td>$= 100$</td>
<td>$= 1/10 * 10000$</td>
</tr>
<tr>
<td></td>
<td>$= 1000$</td>
</tr>
</tbody>
</table>
**Index Expectations**

- Sequential disk reads are faster than random ones
  - Cost ~1-2% random scan = full sequential scan

![Graph showing cost to scan unclustered and clustered indices compared to sequential scans.](image)

- **Proportion of Data Needed**
  - 1-2%
  - 100%

- **Cost to Scan**
  - Unclustered Index Scan
  - Clustered Index Scan
  - Sequential Scan

November 1, 2019
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 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 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 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)

Fine to create indexes on non-numeric data (just not discussed in this class)

Unclustered by default

Order specifies precedence in sorting

Reorders data on disk! (Fails if another clustered index exists)
Index-Based Equijoin

- Assume index exists on the join attribute a of S

```
SELECT * 
FROM R, S 
WHERE R.a = S.a 
```

- **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
  - \( B(R) + T(R)(X*T(S)) = B(R) + T(R)(T(S)/V(a,S)) \)
Index-Based Equijoin

- Assume index exists on the join attribute \( a \) of \( S \)

\[
\text{SELECT } * \text{ FROM } R, S \text{ WHERE } R.a = S.a
\]

### 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 \( R \)
- \( B(R) + T(R)(X*T(S)) = B(R) + T(R)(T(S)/V(a,S)) \)

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