Lecture 21: Hash Tables and Query Execution

Friday, March 3, 2006

Outline

• Hash-tables (13.4)
• Query execution: 15.1 – 15.5
Hash Tables

- Secondary storage hash tables are much like main memory ones
- Recall basics:
  - There are $n$ buckets
  - A hash function $f(k)$ maps a key $k$ to $\{0, 1, \ldots, n-1\}$
  - Store in bucket $f(k)$ a pointer to record with key $k$
- Secondary storage: bucket = block, use overflow blocks when needed

Hash Table Example

- Assume 1 bucket (block) stores 2 keys + pointers
- $h(e) = 0$
- $h(b) = h(f) = 1$
- $h(g) = 2$
- $h(a) = h(c) = 3$

Here: $h(x) = x \mod 4$
Searching in a Hash Table

- Search for a:
- Compute \( h(a) = 3 \)
- Read bucket 3
- 1 disk access

Insertion in Hash Table

- Place in right bucket, if space
- E.g. \( h(d) = 2 \)
Insertion in Hash Table

- Create overflow block, if no space
- E.g. \( h(k) = 1 \)

\[
\begin{array}{c}
0 \\
1 \\
2 \\
3 \\
\end{array}
\begin{array}{c}
e \\
b \\
g \\
a \\
\end{array}
\begin{array}{c}
i \\
f \\
d \\
c \\
\end{array}
\]

- More overflow blocks may be needed

Hash Table Performance

- Excellent, if no overflow blocks
- Degrades considerably when number of keys exceeds the number of buckets (i.e. many overflow blocks).
Extensible Hash Table

• Allows hash table to grow, to avoid performance degradation
• Assume a hash function $h$ that returns numbers in $\{0, \ldots, 2^k - 1\}$
• Start with $n = 2^i \ll 2^k$, only look at first $i$ most significant bits

Extensible Hash Table

• E.g. $i=1$, $n=2^i=2$, $k=4$

• Note: we only look at the first bit (0 or 1)
Insertion in Extensible Hash Table

• Insert 1110

• Now insert 1010

• Need to extend table, split blocks
• i becomes 2
Insertion in Extensible Hash Table

• Now insert 0000, then 0101

• Need to split block
Insertion in Extensible Hash Table

- After splitting the block

Extensible Hash Table

- How many buckets (blocks) do we need to touch after an insertion?

- How many entries in the hash table do we need to touch after an insertion?
Performance Extensible Hash Table

• No overflow blocks: access always one read
• BUT:
  – Extensions can be costly and disruptive
  – After an extension table may no longer fit in memory

Linear Hash Table

• Idea: extend only one entry at a time
• Problem: $n$ no longer a power of 2
• Let $i$ be such that $2^i \leq n < 2^{i+1}$
• After computing $h(k)$, use last $i$ bits:
  – If last $i$ bits represent a number $> n$, change msb from 1 to 0 (get a number $\leq n$)
Linear Hash Table Example

• n=3

Linear Hash Table Example

• Insert 1000: overflow blocks…
Linear Hash Tables

- Extension: independent on overflow blocks
- Extend $n:=n+1$ when average number of records per block exceeds (say) 80%

Linear Hash Table Extension

- From $n=3$ to $n=4$
- Only need to touch one block (which one?)

$n=11$
Linear Hash Table Extension

• From n=3 to n=4 finished

• Extension from n=4 to n=5 (new bit)

• Need to touch every single block (why?)

Summary on Hash Tables

• Alternative index structures:
  – Simpler than B+ trees
  – Faster than B+ trees (when not full)
  – Degrade rapidly (when full)

• Used intensively during query processing
DBMS Architecture

How does a SQL engine work?

- SQL query $\rightarrow$ relational algebra plan
- Relational algebra plan $\rightarrow$ Optimized plan
- Execute each operator of the plan

Architecture of a Database Engine
Relational Algebra

• Formalism for creating new relations from existing ones
• Its place in the big picture:

```plaintext
Declartive query language  ---->  Algebra  ---->  Implementation

SQL, relational calculus

Relational algebra
Relational bag algebra```

Relational Algebra

• Five operators:
  – Union: ∪
  – Difference: -
  – Selection: σ
  – Projection: Π
  – Cartesian Product: ×
• Derived or auxiliary operators:
  – Intersection, complement
  – Joins (natural, equi-join, theta join, semi-join)
  – Renaming: ρ
1. Union and 2. Difference

- R1 ∪ R2
- Example:
  - ActiveEmployees ∪ RetiredEmployees

- R1 – R2
- Example:
  - AllEmployees -- RetiredEmployees

What about intersection?

- It is a derived operator
- R1 ∩ R2 = R1 – (R1 – R2)
- Also expressed as a join (will see later)
- Example
  - UnionizedEmployees ∩ RetiredEmployees
3. Selection

• Returns all tuples which satisfy a condition
• Notation: \( \sigma_c(R) \)
• Examples
  – \( \sigma_{\text{Salary} > 40000} \)(Employee)
  – \( \sigma_{\text{name} = \text{"Smith"}} \)(Employee)
• The condition c can be =, <, ≤, >, ≥, <>
4. Projection

- Eliminates columns, then removes duplicates
- Notation: \( \Pi_{A_1, \ldots, A_n}(R) \)
- Example: project social-security number and names:
  - \( \Pi \text{SSN, Name} (\text{Employee}) \)
  - Output schema: \( \text{Answer(SSN, Name)} \)

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234545</td>
<td>John</td>
<td>200000</td>
</tr>
<tr>
<td>5423341</td>
<td>John</td>
<td>600000</td>
</tr>
<tr>
<td>4352342</td>
<td>John</td>
<td>200000</td>
</tr>
</tbody>
</table>

\( \Pi_{\text{Name,Salary}} (\text{Employee}) \)

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>20000</td>
</tr>
<tr>
<td>John</td>
<td>60000</td>
</tr>
</tbody>
</table>
5. Cartesian Product

- Each tuple in R1 with each tuple in R2
- Notation: R1 \times R2
- Example:
  - Employee \times Dependents
- Very rare in practice; mainly used to express joins

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>EmpSSN</th>
<th>Dname</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>9999999999</td>
<td>9999999999</td>
<td>Emily</td>
</tr>
<tr>
<td>Tony</td>
<td>7777777777</td>
<td>7777777777</td>
<td>Joe</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>EmpSSN</th>
<th>Dname</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>9999999999</td>
<td>9999999999</td>
<td>Emily</td>
</tr>
<tr>
<td>Tony</td>
<td>7777777777</td>
<td>7777777777</td>
<td>Joe</td>
</tr>
<tr>
<td>John</td>
<td>9999999999</td>
<td>9999999999</td>
<td>Emily</td>
</tr>
<tr>
<td>Tony</td>
<td>7777777777</td>
<td>7777777777</td>
<td>Joe</td>
</tr>
</tbody>
</table>
Relational Algebra

- Five operators:
  - Union: $\cup$
  - Difference: $-$
  - Selection: $\sigma$
  - Projection: $\Pi$
  - Cartesian Product: $\times$

- Derived or auxiliary operators:
  - Intersection, complement
  - Joins (natural, equi-join, theta join, semi-join)
  - Renaming: $\rho$

Renaming

- Changes the schema, not the instance
- Notation: $\rho_{B_1,\ldots,B_n}(R)$
- Example:
  - $\rho_{\text{LastName, SocSocNo}}(\text{Employee})$
  - Output schema:
    Answer(\text{LastName, SocSocNo})
Renaming Example

<table>
<thead>
<tr>
<th>Employee</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>SSN</td>
</tr>
<tr>
<td>John</td>
<td>999999999</td>
</tr>
<tr>
<td>Tony</td>
<td>777777777</td>
</tr>
</tbody>
</table>

\[ \rho_{LastName, SocSocNo}(Employee) \]

<table>
<thead>
<tr>
<th>LastName</th>
<th>SocSocNo</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>999999999</td>
</tr>
<tr>
<td>Tony</td>
<td>777777777</td>
</tr>
</tbody>
</table>