Efficient Bulk Deletes in Relational Databases

A. Gärtner¹  A. Kemper¹  D. Kossmann²  B. Zeller¹

¹ Universität Passau  
94030 Passau, Germany

<lastname>@db.fmi.uni-passau.de

² Technische Universität München  
81667 München, Germany

kossmann@in.tum.de
Outline

• Motivation
• Related work
• Idea: traditional horizontal vs. new vertical approach
• Implementation
• Concurrency control and reorganisation
• Benchmark results
• Conclusion and future work
Motivation

• bulk deletes used in:
  • SAP R/3 → Archiving
  • data warehouses → window technique

• today: 500 MB, 3 indices, 15% deleted → 2 h 50 min

• partitioning → drop partition, but:
  • orthogonal deletes:
    • data is partitioned according to orderdate
    • data is deleted according to the orderstatus flag
  • semantic restrictions
    • „delete all orders with orderdate < 1995 but only if the order is fully processed“
Index Structure

- B+-Trees used for indexing
- Leaf pages linked together
  - root
  - inner nodes
  - leaves

- Entries in leaves are sorted according to indexed values
- Record identified by unique Row Identifier (RID)
- RID contains physical address

<table>
<thead>
<tr>
<th>RID:</th>
<th>001</th>
<th>005</th>
<th>017</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileNr</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BlockNr</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SlotNr</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Related Work

- deletion in B⁺-Trees
  - J.Jannink. Implementing deletion in B⁺-Trees. ACM SIGMOD 1994

- bulk loading
Related Work

• Concurrent transactions
  - C.Mohan and F.Levine. ARIES/IM: An efficient and high concurrency index management method using write-ahead logging. ACM SIGMOD 1992

• Pointer join processing
Traditional Horizontal Approach (Record-at-a-time approach)

Table R

<table>
<thead>
<tr>
<th>RID</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
</table>

Index on A

Index on B
Traditional Horizontal Approach
(Record-at-a-time approach)

Table R

<table>
<thead>
<tr>
<th>RID</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
</table>

Index on A

Index on B

disk pages
Traditional Horizontal Approach
(Record-at-a-time approach)

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)
Traditional Horizontal Approach
(Record-at-a-time approach)

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)

Table R

<table>
<thead>
<tr>
<th>RID</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
</table>

Index on A

disk pages

Index on B
Traditional Horizontal Approach
(Record-at-a-time approach)

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)
Traditional Horizontal Approach
(Record-at-a-time approach)

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)
Traditional Horizontal Approach (Record-at-a-time approach)

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)

Table R

<table>
<thead>
<tr>
<th>RID</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0107</td>
<td>4711</td>
<td>700</td>
</tr>
</tbody>
</table>

Index on A

Index on B

disk pages
Traditional Horizontal Approach (Record-at-a-time approach)

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)

Table R

<table>
<thead>
<tr>
<th>RID</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0107</td>
<td>4711</td>
<td>700</td>
</tr>
</tbody>
</table>

Index on A

Index on B
Traditional Horizontal Approach (Record-at-a-time approach)

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)

Table R

<table>
<thead>
<tr>
<th>RID</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0107</td>
<td>4711</td>
<td>700</td>
</tr>
</tbody>
</table>

Index on A

Index on B

disk pages
Traditional Horizontal Approach (Record-at-a-time approach)

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)

<table>
<thead>
<tr>
<th>RID</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0107</td>
<td>4711</td>
<td>700</td>
</tr>
</tbody>
</table>

Index on A

Index on B

disk pages
Traditional Horizontal Approach (Record-at-a-time approach)

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)

Table R

<table>
<thead>
<tr>
<th>RID</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0107</td>
<td>4711</td>
<td>700</td>
</tr>
</tbody>
</table>

Index on A

Index on B

disk pages
Traditional Horizontal Approach (Record-at-a-time approach)

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)

Table R

<table>
<thead>
<tr>
<th>RID</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0107</td>
<td>4711</td>
<td>700</td>
</tr>
</tbody>
</table>

Index on A

Index on B

disk pages
Traditional Horizontal Approach (Record-at-a-time approach)

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)
Traditional Horizontal Approach (Record-at-a-time approach)

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)

Table R

<table>
<thead>
<tr>
<th>RID</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0107</td>
<td>4711</td>
<td>700</td>
</tr>
</tbody>
</table>

Index on A

Index on B

disk pages
Traditional Horizontal Approach
(Record-at-a-time approach)

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)

Table R

<table>
<thead>
<tr>
<th>RID</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0107</td>
<td>4711</td>
<td>700</td>
</tr>
</tbody>
</table>

=> Random I/O on index leaf pages and table pages
New Set-Oriented Vertical Approach

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)
Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)
Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)
New Set-Oriented Vertical Approach

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)

Index on A

sort_A

disk pages

<table>
<thead>
<tr>
<th>RID</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
</table>

Index on B
New Set-Oriented Vertical Approach

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)
New Set-Oriented Vertical Approach

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)
New Set-Oriented Vertical Approach

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)
Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)
New Set-Oriented Vertical Approach

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)
New Set-Oriented Vertical Approach

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)

Index on A

Index on B

disk pages

<table>
<thead>
<tr>
<th>RID</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
</table>

sort_A

sort_RID
Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)
New Set-Oriented Vertical Approach

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)
New Set-Oriented Vertical Approach

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)
New Set-Oriented Vertical Approach

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)
New Set-Oriented Vertical Approach

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)

=> Sequential I/O on index leaf pages and table pages
Implementing the new Approach

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)
Implementing the new Approach

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)
Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)
Reuse of traditional techniques

• Join operator $\rightarrow$ bulk delete operator:

\[
\text{Delete Set D} \quad \text{Index I' / Relation R'}
\]

\[
\text{Index I / Relation R}
\]

• generating bulk delete evaluation plans using existing optimizers
  • $\downarrow$ method
  • $\leftarrow$ order
  • primary $\downarrow$ predicate

• different join techniques, but one argument in place
Example

• Delete from R where R.A in D(A)

• Schema:

```
| RID | A | B | C | ...
|-----|---|---|---|---
|     |   |   |   |   |
```

Index A

Index B

Index C
New Set-Oriented Vertical Approach

Delete from R where R.A in (4711, 5614, 3817, 2918, 5917)
Query Evaluation Plans: Retrieve Index Keys B from R

Diagram:
- D(A) to sort_A
- sort_A to [A]
- [A] to merge_A
- merge_A to [B,RID]
- [B,RID] to sort_B
- sort_B to merge_B,RID
- merge_B,RID to IB(B,RID)
- IB(B,RID) to R(RID,A,B,C,...)
- R(RID,A,B,C,...) to IA(A,RID)
- IA(A,RID) to D(A)
Query Evaluation Plans: Retrieve Index Keys B, C from R

$I_C(C,RID) → \text{merge}_{C,RID} \rightarrow [C,RID] \rightarrow \text{sort}_C \rightarrow \text{merge}_{RID} \rightarrow R(RID,A,B,C,...)$

$I_B(B,RID) → \text{merge}_{B,RID} \rightarrow [B,RID] \rightarrow \text{sort}_B \rightarrow \text{merge}_{RID} \rightarrow R(RID,A,B,C,...)$

$D(A) → \text{sort}_A \rightarrow \text{merge}_A \rightarrow \text{merge}_{C,RID} \rightarrow [C,RID] \rightarrow \text{sort}_C \rightarrow \text{merge}_{RID} \rightarrow R(RID,A,B,C,...)$

$I_A(A,RID) → \text{sort}_{RID} \rightarrow \text{merge}_{RID} \rightarrow [B,RID] \rightarrow \text{sort}_B \rightarrow \text{merge}_{RID} \rightarrow R(RID,A,B,C,...)$
Query Evaluation Plans: RID-Join using range partitioning

Range partitioning on B using DB statistics

[D(A)]

sort\textsubscript{A}

merge \textsubscript{A}

sort\textsubscript{RID}

merge \textsubscript{RID}

[B,RID]

build

hash \textsubscript{RID}

probe

I\textsubscript{B}(B,RID)

I\textsubscript{A}(A,RID)
Query Evaluation Plans: RID-Join on Indices

Build Relation has to fit in memory

Join and update in one pass

$R(RID, A, B, C, \ldots)$

$D(A)$

$#A(A, RID)$

$IB(B, RID)$

$sort_{A}$

$mer$ge $A$

$sort_{RID}$

$[RID]$
Concurrency Control
Concurrency Control

• Take table R and all indices off-line (avoid lock escalation)
Concurrency Control

• Take table R and all indices off-line (avoid lock escalation)

• Process R and all indices with unique constraints
Concurrency Control

• Take table R and all indices off-line (avoid lock escalation)

• Process R and all indices with unique constraints

• Bring R and indices with unique constraints on-line
Concurrency Control

• Take table R and all indices off-line (avoid lock escalation)

• Process R and all indices with unique constraints

• Bring R and indices with unique constraints on-line

• Remaining indices can be processed concurrently using sidefiles/direct propagation

(C.Mohan and I.Narang. Algorithms for creating indexes for very large tables without quiescing updates. ACM SIGMOD 1992.)
Concurrent Control

- Take table R and all indices off-line (avoid lock escalation)
- Process R and all indices with unique constraints
- Bring R and indices with unique constraints on-line
- Remaining indices can be processed concurrently using sidefiles/direct propagation
  (C. Mohan and I. Narang. Algorithms for creating indexes for very large tables without quiescing updates. ACM SIGMOD 1992.)
- Checkpoints: save high water mark
Concurrent Control

- Take table R and all indices off-line (avoid lock escalation)
- Process R and all indices with unique constraints
- Bring R and indices with unique constraints on-line
- Remaining indices can be processed concurrently using sidefiles/direct propagation
  (C. Mohan and I. Narang. Algorithms for creating indexes for very large tables without quiescing updates. ACM SIGMOD 1992.)
- Checkpoints: save high water mark
- Recovery: roll forward to save work already done
Reorganisation


• light version:

• Problem: holes

row movement
Benchmark Environment

• Prototype database
• \( \text{B}^+\)-Trees based on Jan Jannink's implementation
• Table R: 1,000,000 records (500 MB)
• Delete set D
• One index \( \text{I}_A \) on attribute A
• Statement:
  
  \[
  \text{delete from R where R.A in (select D.A from D)}
  \]
• sort merge evaluation plan used
Tests

- Vary number of deleted records
- Vary number of indices
- Vary height of indices
- Vary size of memory
- Clustered table

Diagrams:
- not sorted/trad:  Delete set not sorted, traditional approach
- sorted/trad     Delete set sorted, traditional approach
- not sorted/bulk Delete set not sorted, vertical approach
Vary Number of Deleted Records

![Graph showing the relationship between deleted tuples and time for different methods]

- **not sorted trad.**
- **sorted trad.**
- **not sorted bulk**

- Time (min):
  - 0
  - 20
  - 40
  - 60
  - 80
  - 100
  - 120
  - 140

- Deleted tuples (% of tuples):
  - 5
  - 10
  - 15
  - 20
Vary Number of Indices

Number of indices (15% deleted)

Time (min)

- not sorted/trad.
- sorted/trad.
- not sorted/bulk
- drop/create
Vary Height of Indices

15% deleted

<table>
<thead>
<tr>
<th></th>
<th>index height 3 (min)</th>
<th>index height 4 (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>not sorted/bulk</td>
<td>25</td>
<td>27</td>
</tr>
<tr>
<td>sorted/trad</td>
<td>65</td>
<td>81</td>
</tr>
<tr>
<td>not sorted/trad</td>
<td>102</td>
<td>136</td>
</tr>
</tbody>
</table>
Vary Size of Memory

![Graph showing the relationship between memory size and time.]
Clustered Table

The graph shows the relationship between the percentage of deleted tuples and the time (in minutes) it takes to sort and cluster (sorted/trad/clust), sort and uncluster (sorted/trad/unclust), bulk clustering (bulk/clust), and not sorting and clustering (not sorted/trad/clust).
Conclusion

• New set-oriented vertical approach
• Implementation using evaluation plans
  • bulk delete operator
  • retrieve index keys from R
  • RID join using hashing
• Concurrency control and reorganisation
• Benchmark results
  → new approach outperforms trad. approach
• Future Work: generalize the approach to hash tables, R- trees, grid files, bitmap indices