Partitioned B-trees
A user's guide

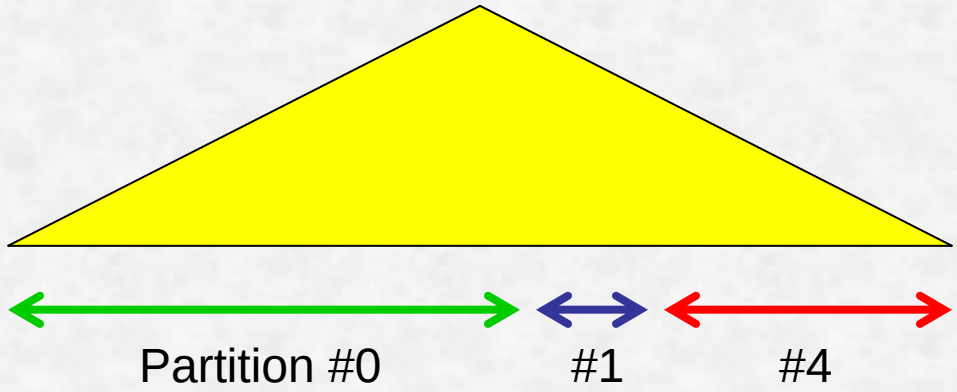
Goetz Graefe
Microsoft SQL Server

What are partitioned B-trees?

- Ordinary B-trees
  - Single- or multi-column, numbers or strings
  - Computed columns, hash values, Z-values, …
- Additional leading key column
  - Used for data management, not in applications
  - Almost always the same value in all rows
  - Temporarily multiple values
  - Online reorganization

Partitioned B-trees

- Artificial leading key column defines partitions
  - Partitions come & go by row insert & delete
  - No schema changes, no table locks
  - No query & plan recompilations
- Online reorganization = external merge sort

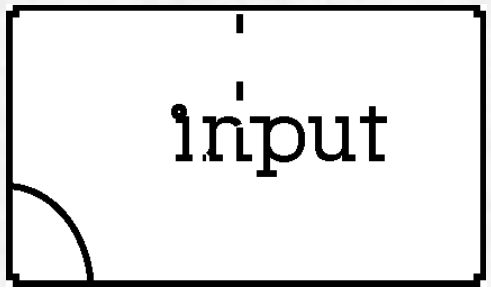


How to build partitioned B-trees

- Vendor, e.g., Microsoft SQL Server
  - Additional column can remain hidden
  - One additional column for each index
  - “Tricks” hide storage & comparison costs
- DB administrator or application developer
  - Additional column is visible
  - One additional column per table, for all indexes

A small case study

- Data warehouse environment
  - Fact table with 100M of rows in 1M pages
  - Clustered index on a column other than time
- Desired operations
  - Delete the oldest 1M rows
    - One week in two years, one day in a quarter, etc.
  - Insert similar amount of new data
    - 1% of records affects ~1 record in every index leaf



Traditional insert processing

- Row-by-row execution
  - In random order with respect to the index order
- Maintain all indexes for one row at a time
- Lots of random I/O: 1M reads + 1M writes
  - At 1,000 I/Os per second: 2,000 sec ≈ ½ hour
- Lots of (exclusive) locks held for a long time
  - Is the data analysis application still online?

Bulk insert processing

- Index-by-index optimized update plan
- Sort the change set for each index
  - Cost-based choice since SQL Server 7.0
- Apply changes in index sort order
  - "Merge" changes into B-tree
  - Each leaf is read & written once
  - Sequential I/O is 2-4 faster, or ≈ ¼ hour

Drop & rebuild indexes

- Drop is very fast: ≈ 0 minutes
- Append new data to heap file
  - 1M rows ≈ 10K pages ≈ 2½ - 5 seconds
- Rebuild all indexes
  - 1.01M pages read & written twice
  - 4M pages I/O ≈ 1,000 sec ≈ ¼ hour
- Entire fact table is offline for the entire time
  - It cannot be queried without its indexes
  - Effectively, the entire data warehouse is offline

Insert into partitioned B-trees

- Presume a single partition at the start
- Ensure single partition at the end
- *Insert data into a new partition*
  - Use a new value in the artificial leading key
  - Append sorted data into entirely new pages
  - 10K pages ≈ 2½ - 5 seconds
- Index remains operational
  - Even if multiple non-clustered indexes exist
  - Only the new data is locked

Merging old & new partitions

- Old data remain in old partition
  - Not locked, not modified, fully operational
  - Permits queries and updates
- After insert (append) is complete
  - All data is immediately indexed & searchable!
  - Small loss of search efficiency
  - Online reorganization moves a few rows at a time
    - External merge sort in (small) key ranges
    - Frequent transaction commits

Some sample SQL

```
Create table t (a int, b float, c varchar(10), …)
Alter table t add column x int check (value >= 0)
Create clustered index … on t (x, a, …)
Insert t (x, a, b, c) select 1, r.a, r.b, r.c from r …
Insert t (x, a, b, c) select 2, r.a, r.b, r.c from r …
…
While exists (select * from t where x in (1, 2, 3))
Begin
   Update t set x = 0 where x in (1, 2, 3) and
         a <= … /* ~1% of key range */
End
```

Related readings

"Sorting and Indexing with Partitioned B-trees"

Conf. on Innovative Data Systems Research

www-db.cs.wisc.edu/cidr

"Efficient Search of Multi-Dimensional B-trees"

VLDB Conf. 1995 (Tandem/HP)

www.vldb.org