第五章 索引

引言

该部分介绍索引技术的基本原理、发展历史以及现阶段最新研究成果

5.1 索引基本原理

该部分主要介绍几种索引方式

5.2 索引相关技术

该部分会介绍索引已有的相关技术，包括比较成熟的应用在各个数据库中的，也包括一些最新的研究成果原理介绍

包括常见的 B-Tree 索引、Hash 索引、Bitmap 索引、Gist 索引、GIN 索引。还有一些最新的研究成果。

Order Indexes

Order Indexes: supporting highly dynamic hierarchical data in relational main-memory database systems VLDB 2017

Order Index 提出主要是要解决具有层次结构的数据在关系数据库中的更新问题。

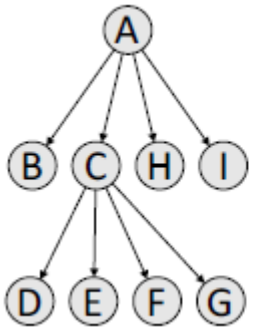
我们知道具有层次结构的数据，在关系型数据库中存储的时候需要存储其等级关系。在查询的时候我们也是需要可以得到一个很快的响应时间，那么在对其进行更新的时候我们不仅需要修改数据还需要

修改对应变更的关系，也需要可以进行快速的更新。

Challenges

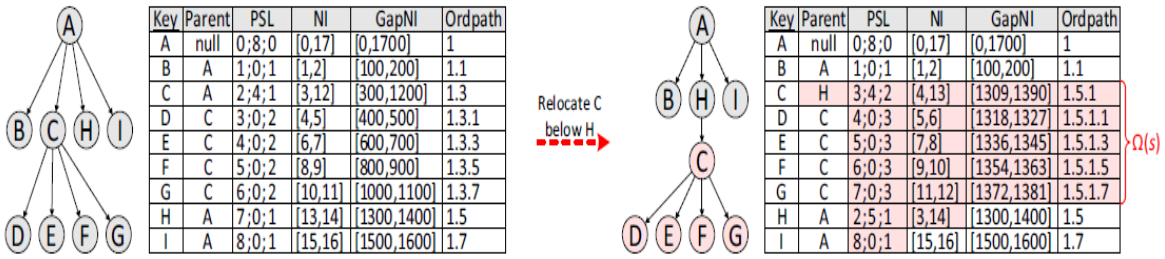
Three problems:

P1 Lack of query capabilities：一些索引方案的确可以支持便利更新，但是并不支持查询操作，即使是一些最基本的查询也无法满足，这使得它们无法满足我们这里的用例。一个典型的例子就是临近列表模型（adjacency list model），在 SQL 中常见的编码这种层次数据是通过存储父节点的键值（如下图所示）。这种模型甚至不能够有效的实现祖先子孙关系的处理。



Key	Parent	PSL	NI	GapNI	Ordpath
A	null	0;8;0	[0,17]	[0,1700]	1
B	A	1;0;1	[1,2]	[100,200]	1.1
C	A	2;4;1	[3,12]	[300,1200]	1.3
D	C	3;0;2	[4,5]	[400,500]	1.3.1
E	C	4;0;2	[6,7]	[600,700]	1.3.3
F	C	5;0;2	[8,9]	[800,900]	1.3.5
G	C	6;0;2	[10,11]	[1000,1100]	1.3.7
H	A	7;0;1	[13,14]	[1300,1400]	1.5
I	A	8;0;1	[15,16]	[1500,1600]	1.7

P2 Lack of complex update capabilities：大量的用例需要索引方案支持大量有效的更新操作。但是多数现有的方案都是基于叶节点的更新，并不支持更加复杂的更新操作。考虑子树重分配（subtree relocation）问题，传统的邻接表虽然支持此更新操作，但是操作代价太大，因为需要重新定位子树中的所有节点。



Key	Parent	PSL	NI	GapNI	Ordpath
A	null	0;8;0	[0,17]	[0,1700]	1
B	A	1;0;1	[1,2]	[100,200]	1.1
C	A	2;4;1	[3,12]	[300,1200]	1.3
D	C	3;0;2	[4,5]	[400,500]	1.3.1
E	C	4;0;2	[6,7]	[600,700]	1.3.3
F	C	5;0;2	[8,9]	[800,900]	1.3.5
G	C	6;0;2	[10,11]	[1000,1100]	1.3.7
H	A	7;0;1	[13,14]	[1300,1400]	1.5
I	A	8;0;1	[15,16]	[1500,1600]	1.7

Key	Parent	PSL	NI	GapNI	Ordpath
A	null	0;8;0	[0,17]	[0,1700]	1
B	A	1;0;1	[1,2]	[100,200]	1.1
C	H	3;4;2	[4,13]	[1309,1390]	1.5.1
D	C	4;0;3	[5,6]	[1318,1327]	1.5.1.1
E	C	5;0;3	[7,8]	[1336,1345]	1.5.1.3
F	C	6;0;3	[9,10]	[1354,1363]	1.5.1.5
G	C	7;0;3	[11,12]	[1372,1381]	1.5.1.7
H	A	2;5;1	[3,14]	[1300,1400]	1.5
I	A	8;0;1	[15,16]	[1500,1600]	1.7

Fig. 1 Various labeling schemes (*left*); labels that need to be changed when subtree C is relocated (*right*) (color figure online)

P3 Vulnerability to skewed updates: 一些动态标签方案在面临不均匀的更新时会崩溃。例如在同一个位置重复插入数据点，而这样的操作在现实中可能会经常性出现。固定长度标签方案可能会陷入重标签错误，而可变长标签方案可能会因为过度增长的标签量导致查询效率和内存效率衰减。

Query capabilities

BINARY PREDICATES	
is_descendant(a, b)	whether a is a descendant of b
is_child(a, b)	whether a is a child of b
is_before_pre(a, b)	whether a precedes b in a pre-order traversal
is_before_post(a, b)	whether a precedes b in a post-order traversal
NODE PROPERTIES	
level(a)	number of edges on the path from a root to a
is_root(a)	whether a is a root node
is_leaf(a)	whether a is a leaf node, i. e., has no children
INDEX & TABLE ACCESS	
find(a)	a cursor c to node a
rowid(c)	the row id corresponding to the node at c
label(c)	the label of the node at c
TRAVERSAL	
next_pre(c)	a cursor to the next node in pre-order
next_post(c)	a cursor to the next node in post-order
next_sibling(c)	a cursor to the next sibling
next_following(c)	a cursor to the next node in pre-order that is not a descendant of c

a, b: node labels, c: index cursor

Fig. 2 Essential query primitives on hierarchies

ORDINAL PROPERTIES	
pre_rank(a)	a's rank in a pre-order traversal
post_rank(a)	a's rank in a post-order traversal
subtree_size(a)	number of nodes in the subtree rooted in a
range_size$([a, b])$	number of nodes in all subtrees rooted in $[a, b]$
ORDINAL ACCESS	
select_pre(n)	a cursor to the n-th node in pre-order
select_post(n)	a cursor to the n-th node in post-order

Fig. 3 Ordinal query primitives

Update capabilities

BULK UPDATES	*(re)build the hierarchy as a whole*
bulk_build(T)	builds the hierarchy from tree representation T
LEAF UPDATES	*alter a single leaf node*
delete_leaf(a)	deletes a leaf node a
insert_leaf(a, p)	inserts the new leaf node a at position p
relocate_leaf(a, p)	relocates a leaf node a to position p
SUBTREE UPDATES	*alter a subtree*
delete_subtree(a)	deletes the subtree rooted in a
insert_subtree(a, p)	inserts the new subtree rooted in a at position p
relocate_subtree(a, p)	relocates the subtree rooted in a to position p
RANGE UPDATES	*alter subtrees rooted in a range of siblings*
delete_range($[a, b]$)	deletes all subtrees rooted in range $[a, b]$
insert_range($[a, b], p$)	inserts all subtrees rooted in range $[a, b]$ at position p
relocate_range($[a, b], p$)	relocates all subtrees rooted in range $[a, b]$ to position p
INNER UPDATES	*alter an inner node*
delete_inner(a)	deletes node a and puts its children into its place below its former parent
insert_inner($a, [b, c]$)	inserts the new node a in place of range $[b, c]$ and makes $[b, c]$ children of a
relocate_inner($a, [b, c]$)	puts children of a into a's place and moves a to the place of $[b, c]$, which become children of a

Fig. 4 Update operations on hierarchies

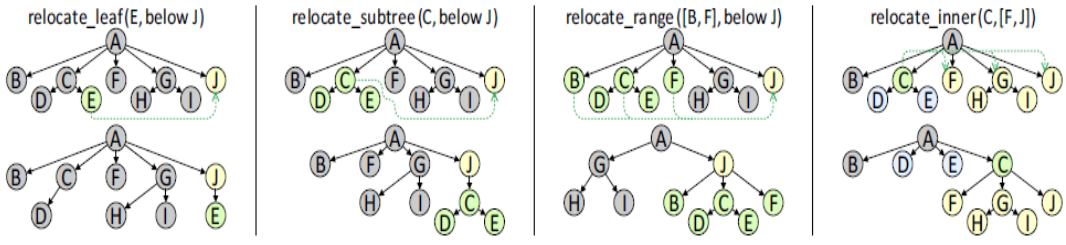


Fig. 5 Various classes of relocation updates on an example hierarchy: before (*top*) and after (*bottom*) (color figure online)

Order Index

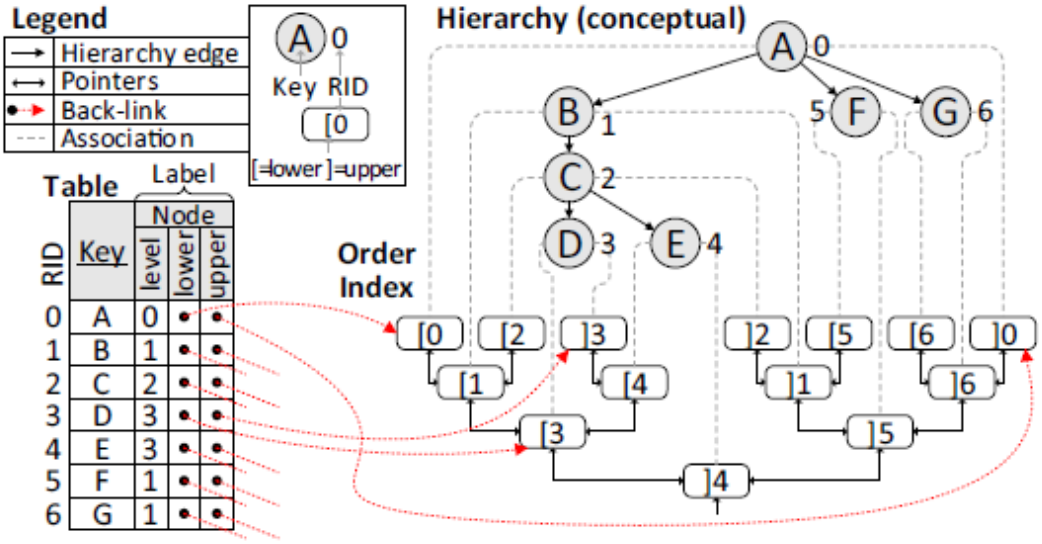


Fig. 9 A hierarchy with Order Index (AO-Tree) (color figure online)

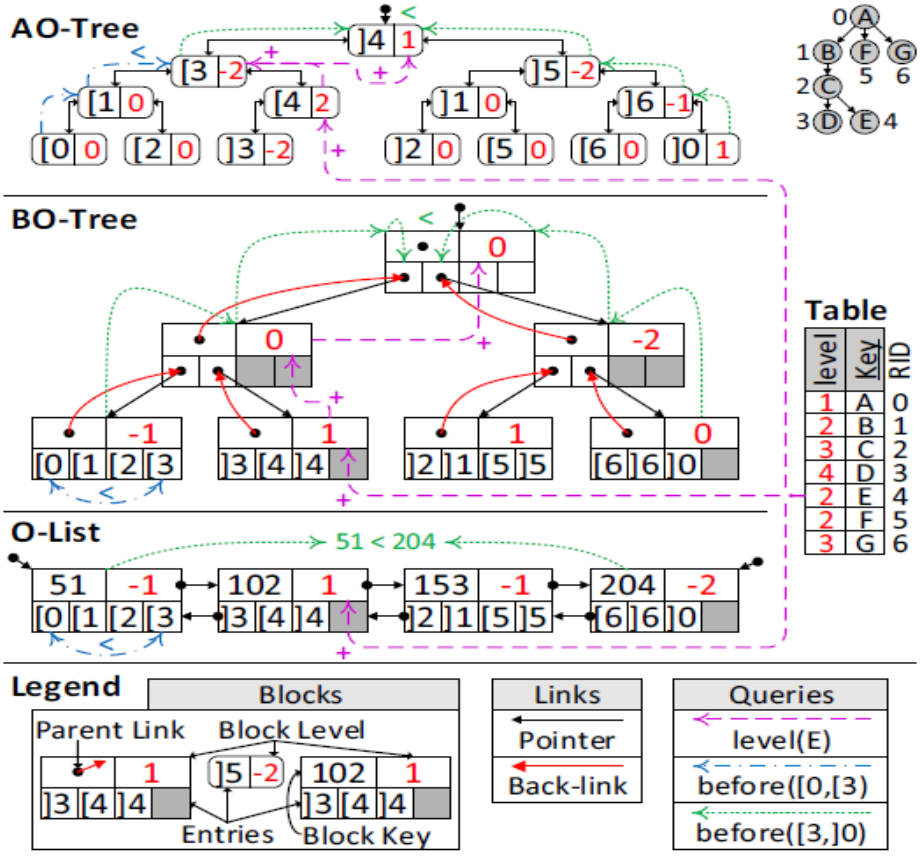


Fig. 11 Query evaluation in the Order Indexes (color figure online)

Update-buffer-based indexing

Read/write-optimized tree indexing for solid-state drives

Peiquan Jin[1,2] · Chengcheng Yang[1,2] · Christian S. Jensen[3] · Puyuan Yang[1,2] · Lihua Yue[1,2]

VLDB 2016

基于 SSD 的存储介质和内存数据库的出现对传统的索引技术提出了新的挑战。SSD 有着高速的读效率但是写效率较低，以及“out-of-place update”的特性。传统的 B-树索引和 hash 索引主要是针对机械硬盘，需要重新设计来适应 SSD 的这些特性。该工作针对传统的 Tree 做了优化来提高索引效率。

The Case for Learned Index Structures

该部分作为未来工作展望给出

5.3 索引技术在 Greenplum 中的实现

该部分会具体介绍 Greenplum 中支持的几种索引技术

5.4 索引实验部分

该部分会简要介绍在 Greenplum 中使用索引进行实验部分

5.5 参考文献及进一步阅读材料

该部分会列举一些前面涉及到的参考文献，也会给出一些推荐阅读文献

5.6 本章小结

该部分对本章进行总结