**1. B 树**

实现的功能

1. 打开/关闭 chidb 文件
2. 从文件从读取 B 树的节点
3. 创建 B 树节点和将 B 树节点写入文件
4. 获取和插入 Cell 到 B 树节点中
5. 在 B 树中查找值
6. 向 B 树中插入 Cell

### 1.1 chidb 文件格式

#### 1.1.1 逻辑组织

chidb文件包含以下内容：

* 文件头。
* 0个或更多表。
* 0个或多个索引。

数据库文件中的每个表都存储为B + -Tree（我们将它们简称为表B-Trees）。该树的条目是数据库记录（对应于表中一行的值的集合）。每个条目的键将是其主键。由于表是B + -Tree，因此内部节点不包含条目，而是用于导航树。

数据库文件中的每个索引都存储为B树（我们将其称为索引B树）。假设我们有关系 R(pk,…,ik,…) 哪里 pk 是主键，并且 ik 是我们要在其上创建索引的属性，索引B-Tree中的每个条目将是一个 (k1,k2) 一对，哪里 k1 是一个值 ik 和 k2 是主键的值（pk）在记录中 ik=k1。

#### 1.1.2 物理组织

chidb文件分为若干大小的页面PageSize，从1开始编号。每个页面用于存储表或索引B-Tree节点。

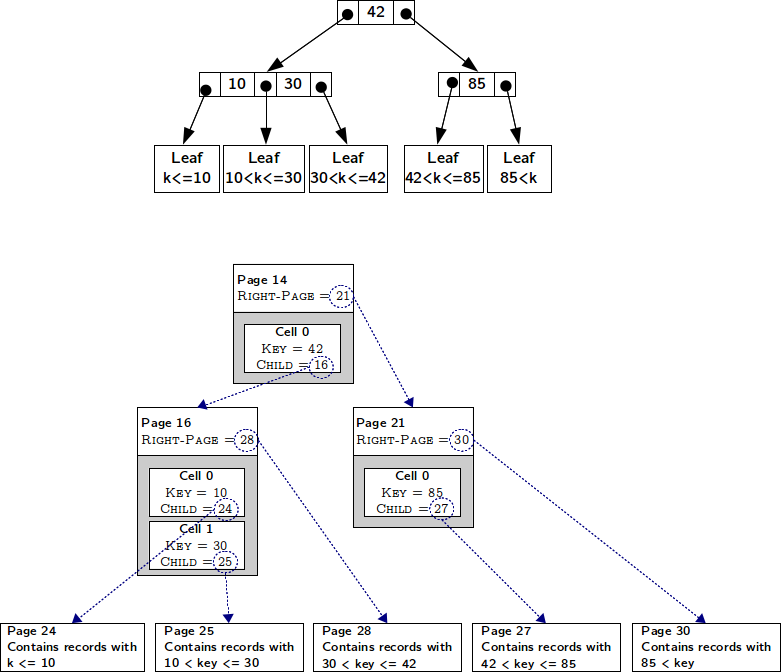
页面包含带有页面相关元数据的页面标题，例如页面类型。标头未使用的空间可用于存储 单元，这些单元用于存储B树条目：

* 叶表单元格：⟨Key,DBRecord⟩，在哪里 DBRecord 是数据库记录，并且 Key 是它的主键。
* 内部表格单元：⟨Key,ChildPage⟩，在哪里 ChildPage 是包含键小于或等于的条目的页面的编号 Key。
* 叶索引单元格：⟨KeyIdx,KeyPk⟩，在哪里 KeyIdx 和 KeyPk 是 k1 和 k2，如先前定义。
* 内部索引单元格：⟨KeyIdx,KeyPk,ChildPage⟩，在哪里 KeyIdx 和 KeyPk 如上定义，并且 ChildPage 是包含键小于的条目的页面数 KeyIdx。

数据库中的页面1很特殊，因为文件头使用了它的前100个字节。因此，存储在页面1中的B树节点只能使用 (PageSize−100) 个字节。

B树节点必须具有 n 键和 n+1 指针。但是，使用单元格，我们只能存储n指针。给定一个节点B，则需要一个额外的指针来存储包含该节点的页面号 B′ 键大于所有的键 B。这个额外的指针存储在页面标题中，称为 RightPage。

表B树的逻辑和物理视图如图 Figure 1 所示。



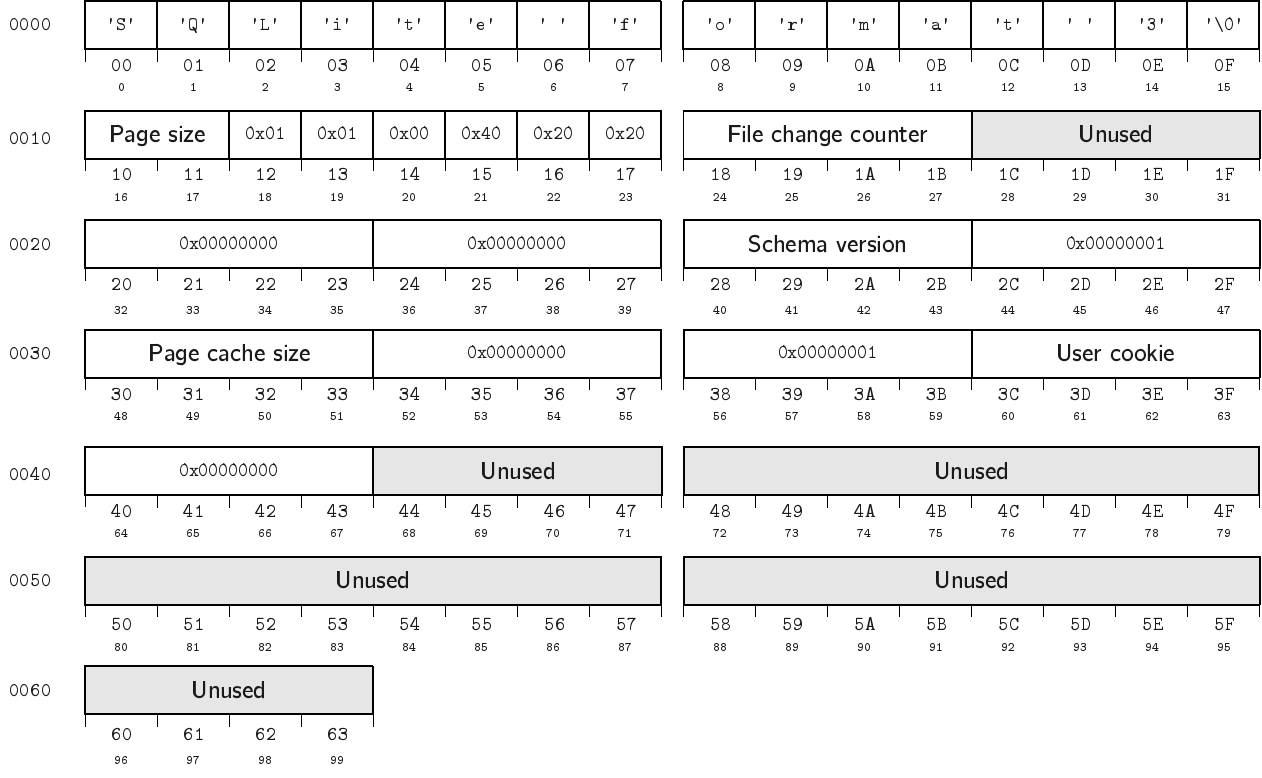
Figure

#### 1.1.3 数据类型

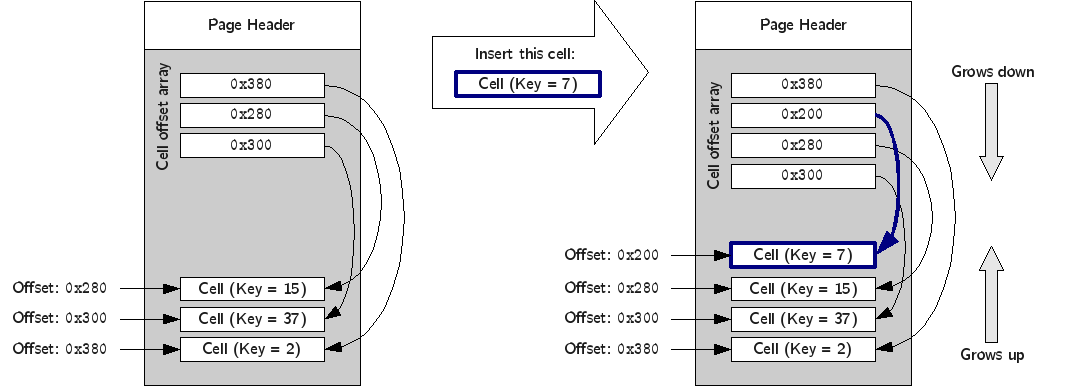


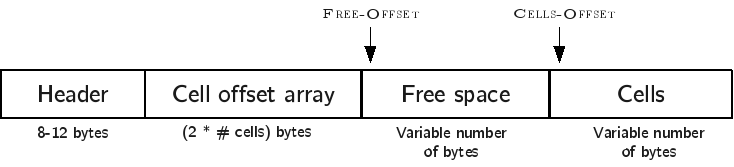
#### 1.1.4 文件头

chidb文件的前100个字节包含带有该文件元数据的标头。该文件头使用与SQLite相同的格式，并且由于chidb中不支持许多SQLite功能，因此大多数头都包含常量值。标头的布局如下图所示。



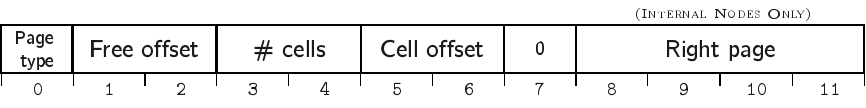
#### 1.1.5 表页





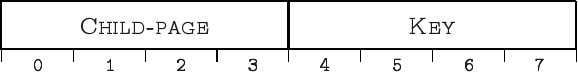
* 在页头位于页面的顶部，并包含有关页面的元数据。页头的确切内容将在后面说明。
* 所述Cells偏移数组在页头之后的位置。每个条目都存储为一个uint16。因此，Cells偏移数组的长度取决于页面中单元格的数量。
* Cells位于页面的结尾。
* Cells偏移数组与Cells之间的空间是可用空间 ，Cells偏移数组可以增长（向下），Cells可以增长（向上）。

下图和表格总结了页头的布局：

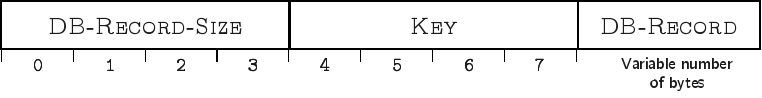


#### 1.1.6 Table Cell

下图和表中指定了内部表格单元的布局：



下图和表中指定了叶表单元格的布局：

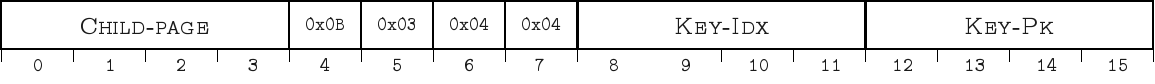


#### 1.1.7 索引

索引B树非常类似于表B树，因此我们解释过的重新映射表B树的大部分内容都适用于索引。主要区别如下：

* PageType 字段必须设置为适当的值（0x02对于内部页面和0x0A叶子页面）
* 表存储为B + -Tree（记录仅存储在叶节点中）时，索引存储为B-Tree（记录存储在所有级别）。但是，索引不存储数据库记录，而是⟨KeyIdx,KeyPk⟩对

下图和表中指定了内部索引单元的布局：



下图和表中指定了叶索引单元的布局：

#### å¶ç»èï¼ææ°ï¼1.2.1 打开/关闭 chidb 文件

/\*\* 打开一个 B 树文件

\*

\* 这个函数打开一个数据库文件并且验证其文件头是否正确.

\* 如果文件是空的(或者文件不存在), 则

\* 1) 通过默认的 page size 初始化文件头

\* 2) 在页 1 上创建一个空的表叶节点

\*/

int chidb\_Btree\_open(const char \*filename, chidb \*db, BTree \*\*bt);

/\*\* 关闭一个 B 树文件

\*

\* 这个函数关闭一个数据库文件, 释放内存中的资源, 比如 pager

\*/

int chidb\_Btree\_close(BTree \*bt);

#### 1.2.2 从文件从读取 B 树的节点

/\*\* 从硬盘中加载一个 B 树结点

\*

\* 从硬盘中读取一个 B 树结点. 所有关于结点的信息都被存储在 BTreeNode 结构体中.

\* 任何影响在 BTreeNode 变量上的改变直到 chidb\_Btree\_writeNode 被调用才会

\* 在数据库中起作用.

\*/

int chidb\_Btree\_getNodeByPage(BTree \*bt, npage\_t npage, BTreeNode \*\*node);

/\*\* 释放分配给 B 树结点的内存

\*

\* 释放分配给 B 树结点的内存, 以及内存中存储的页.

\*/

int chidb\_Btree\_freeMemNode(BTree \*bt, BTreeNode \*btn);

#### 1.2.3 创建 B 树节点和将 B 树节点写入文件

/\*\* 创建一个新的B树节点

\*

\* 在文件中分配一个新页面，并将其初始化为B-Tree节点。

\*/

int chidb\_Btree\_newNode(BTree \*bt, npage\_t \*npage, uint8\_t type);

/\*\* 初始化B树节点

\*

\* 初始化数据库页面以包含一个空的B-Tree节点。假定 Page 已存在，并且已经由 Pager 分配。

\*/

int chidb\_Btree\_initEmptyNode(BTree \*bt, npage\_t npage, uint8\_t type);

/\*\* 将内存中的B-Tree节点写入磁盘

\*

\* 将内存中的B-Tree节点写入磁盘。为此，我们需要根据chidb页面格式更新内存页面。

\* 由于 Cell 偏移数组和单元格本身是直接在页面上修改的,

\* 因此唯一要做的就是将 “type”，“free\_offset”，

\* “n\_cells”，“cells\_offset”

\* 和 “right\_page” 的值存储在内存页。

\*/

int chidb\_Btree\_writeNode(BTree \*bt, BTreeNode \*node);

#### 1.2.4 获取和插入 Cell 到 B 树节点中

/\*\* 读取 Cell 的内容

\*

\* 从BTreeNode读取单元格的内容，并将其存储在BTreeCell中。 这涉及以下内容：

\* 1.找出所需 Cell 的偏移量。

\* 2.从内存页面中读取 Cell，然后解析其内容。

\*/

int chidb\_Btree\_getCell(BTreeNode \*btn, ncell\_t ncell, BTreeCell \*cell);

/\*\* 将新 Cell 插入 B 树节点

\*

\* 在指定位置ncell处将新单元格插入B树节点。这涉及以下内容：

\* 1.将单元格添加到单元格区域的顶部

\* 2.修改BTreeNode中的cells\_offset以反映单元区域中的增长

\* 3.修改 cells 偏移数组，以使位置 >= ncell 中的所有值在数组中向后移动一个位置

\* 然后，将位置ncell的值设置为新添加的 Cell 的偏移量

\*/

int chidb\_Btree\_insertCell(BTreeNode \*btn, ncell\_t ncell, BTreeCell \*cell);

#### 1.2.5 在 B 树中查找值

/\*\* 在表 B 树中查找条目

\*

\* 在表B-Tree中查找与给定键关联的数据

\*/

int chidb\_Btree\_find(BTree \*bt, npage\_t nroot, key\_t key, uint8\_t \*\*data, uint16\_t \*size);

#### 1.2.6 向 B 树中插入 Cell

/\*\* 将条目插入表 B 树

\*

\* 它需要一个键和数据，并创建一个BTreeCell，可以将其传递给chidb\_Btree\_insert。

\*/

int chidb\_Btree\_insertInTable(BTree \*bt, npage\_t nroot,

key\_t key, uint8\_t \*data, uint16\_t size);

/\*\* 将条目插入索引 B 树

\*

\* 它使用一个KeyIdx和一个KeyPk，并创建一个BTreeCell，可以将其传递给chidb\_Btree\_insert。

\*/

int chidb\_Btree\_insertInIndex(BTree \*bt, npage\_t nroot, key\_t keyIdx, key\_t keyPk);

/\*\* 将BTreeCell插入B树

\*

\* chidb\_Btree\_insert和chidb\_Btree\_insertNonFull函数

\* 负责将新条目插入B树, chidb\_Btree\_insertNonFull 实际执行插入

\* chidb\_Btree\_insert，首先检查根是否必须拆分（拆分操作不同于拆分其他任何节点）

\* 如果是这样，则在调用chidb\_Btree\_insertNonFull之前调用chidb\_Btree\_split

\*/

int chidb\_Btree\_insert(BTree \*bt, npage\_t nroot, BTreeCell \*btc);

/\*\* 将BTreeCell插入有空闲空间的B-Tree节点

\*

\* chidb\_Btree\_insertNonFull将BTreeCell插入到一个

\* 假设未满（即不需要拆分）的结点上。

\* 如果节点是叶节点，根据其键的位置将单元格直接添加到适当的位置

\* 如果该节点是内部节点，

\* 则函数将确定必须将其插入哪个子节点，

\* 并且在该子节点上递归调用自身。

\* 但是，在这样做之前它将检查子节点是否已满。

\* 如果是这样，则必须先将其拆分。

\*/

int chidb\_Btree\_insertNonFull(BTree \*bt, npage\_t npage, BTreeCell \*btc);

/\*\* 切分 B 树结点

\*

\* 拆分 B 树节点 N. 这涉及以下内容：

\* - 在 N 中找到处于中间位置的 Cell

\* - 创建一个新的 B 树节点 M

\* - 将中间位置 Cell 之前的单元格移至 M（如果该单元格是表格叶单元格，那么中间位置的 Cell 也将移动）

\* - 使用中键和 M 的页码将一个单元格提升到父结点中。

\*/

int chidb\_Btree\_split(BTree \*bt, npage\_t npage\_parent, npage\_t npage\_child,

ncell\_t parent\_cell, npage\_t \*npage\_child2);