Index B-tree Page - type 0x07

As described above for the Index Root Page (type 0x06) each index defined for a table has a root page from which the index data can be read etc. The Index Root Page field irt_root points to the first page (the apex page) in the index. That page will be a type 0x07 Index B-Tree Page, as will all the other pages that make up this index.

Indices do not share pages. Each index has its own range of dedicated pages in the database. Pages are linked to the previous and next pages making up this index.

B-tree header

The C code representation of an ODS 11 index b-tree page is:

```
struct btree_page
{
   pag btr_header;
   SLONG btr_sibling;
   SLONG btr_left_sibling;
   SLONG btr_prefix_total;
   USHORT btr_relation;
   USHORT btr_length;
   UCHAR btr_id;
   UCHAR btr_level;
};
```

Btr_header: The page starts off with a standard page header. The pag_flags byte is used on these pages. The bits used and why are:

- *Bit 0*: set means do not garbage collect this page.
- *Bit 1*: set means this page is not propogated upwards.
- *Bit 3*: set means that this page/bucket is part of a descending index.
- *Bit 4*: set means that non-leaf nodes will contain record number information.
- *Bit 5*: set means that large keys are permitted/used.
- *Bit 6*: set means that the page contains index jump nodes.

Btr_sibling: Four bytes, signed. Bytes 0x10 - 0x13 on the page. This is the page number of the next page of this index. The values on the next page are higher than all of those on this page. A value of zero here indicates that this is the final page in the index.

Btr_left_sibling: Four bytes, signed. Bytes 0x14 - 0x17 on the page. This is the page number of the previous page of this index. The values on the previous page are lower than all of those on this page. A value of zero here indicates that this is the first page in the index.

Btr_prefix_total: Four bytes, signed. Bytes 0x18 - 0x1b on the page. The sum of all the bytes saved on this page by using prefix compression.

Btr_relation: Two bytes, unsigned. Bytes 0x1c and 0x1d on the page. The relation ID (RDB$RELATION_ID in RDB$RELATIONS) for the table that this index applies to.

Btr_length: Two bytes, unsigned. Bytes 0x1e and 0x1f on the page. The number of bytes used, for data, on this page. Acts as an offset to the first unused byte on the page.

Btr_id: One byte, unsigned. Byte 0x20 on the page. The index ID (RDB$INDEX_ID in RDB$INDICES) for this index.

Btr_level: One byte, unsigned. Byte 0x21 on the page. The index level. Level zero indicates a leaf node.

back to top of page

Index Jump Info

Following on from the above, at byte 0x22 on the page, is an *Index Jump Info* structure. This is defined as follows:

```
struct IndexJumpInfo
{
   USHORT firstNodeOffset;
   USHORT jumpAreaSize;
   UCHAR  jumpers;
};
```

FirstNodeOffset: Two bytes, unsigned. Offset 0x00 in the structure. This is the offset, in bytes, to the first of the Index Nodes (see below) on this page.

JumpAreaSize: Two bytes, unsigned. Offset 0x02 in the structure. The value here is the number of bytes left to be used before we have to create a new jump node.

Jumpers: One byte, unsigned. Offset 0x05 in the structure. The running total of the current number of Jump Nodes on this page. There can be a maximum of 255 Index Jump Nodes on a page.

back to top of page

Index Jump Nodes

The Index Jump Info structure described above is followed by zero or more Index Jump Nodes. The number to be found is determined by the jumpers value in the Index Jump Info structure. Index Jump Nodes are defined as follows:

```
struct IndexJumpNode
{
   UCHAR* nodePointer; // pointer to where this node can be read from the
```

```
page
   USHORT prefix; // length of prefix against previous jump node
   USHORT length; // length of data in jump node (together with prefix this
is prefix
   USHORT offset; // offset to node in page
   UCHAR* data; // Data can be read from here
};
```

nodePointer:

Prefix:

Length:

Offset:

Data:

back to top of page

Index Nodes

Btr_nodes: Index nodes are described below and are used to hold the data for one entry in this index. The C code representation of an entry in the array is:

```
struct btree_nod
{
   UCHAR *nodePointer;
   USHORT btn_prefix;
   USHORT btn_length;
   SLONG pageNumber;
   UCHAR *data;
   RecordNumber recordNumber;
   bool isEndBucket;
   bool isEndLevel;
};
```

NodePointer: ????????? Pointer to where this node can be read from the page.

Btn_prefix: ????????? Size of the compressed prefix.

Btn_length: ????????? Length of the data in the node.

PageNumber: ????????? Page number.

Data: ????????? Data can be read from here.

RecordNumber: ????????? Record number. (A recordNumber is a C++ Class BTW - see jrd/recordnumber.h)

IsEndBucket: ????????? Is this the end of the bucket?

IsEndLevel: ????????? Is this the end of the index at this level?

back to top of page

Index Data

Btn_data: ??????????????????