Multiple and Secondary Indexing

We've seen how **primary keys** can be used to uniquely identify records:

- unique, canonical, dataless, unchanging
- are sorted in an index
- searching the index for a primary key yields a RRN (fixed-length) of byte offset (variable-length)

Therefore, we might associate a primary key for each song record. But what if we wanted to index by artist?

We could use a **secondary key index**, sorted by a particular secondary key, with each index pointing to ...

a collection of records containing the secondary key. But how do we refer to the records in the collection?

Could point directly to the RRN or byte offset of the record. What is the problem with this approach?

Instead, we point to primary keys.

Maintaining a Secondary Key Index

What happens if we add a record?

- since files are entry sequenced, we can add a record at end of file or in some available space record, using our compaction strategies.

- adding a record means adding its primary key to the (sorted) primary key index as well as any relevant secondary key indexes.

What happens if we delete a record?

- primary key is left in primary key index, with entry flagged to indicate deleted record.

- what about the secondary keys?

The Structure of a Secondary Key Index

We spoke earlier of a "collection of primary keys" pointed to by a secondary key. How should this collection be stored?

Could set aside an array of space to hold a fixed number of primary keys. What is wrong with this?

internal fragmentation, upper bound limits. How do we avoid internal fragmentation?

using linked lists or inverted lists. How do we store these linked lists?

right in the file! - we saw this before in our compaction algorithms for fixed-length records. What is the cost of such a mechanism?

more seeks.