Lecture 12: Indexing

Announcements

1. Great work on the midterm!

2. You're halfway done!

3. This half of the class will be more relaxed
   - Project Part 2 due next Wednesday
   - Project Part 3 due on November 22nd (before Thanksgiving ☺)

4. What do you want to do on November 22nd?

Lecture 12: Indexing

What you will learn about in this section

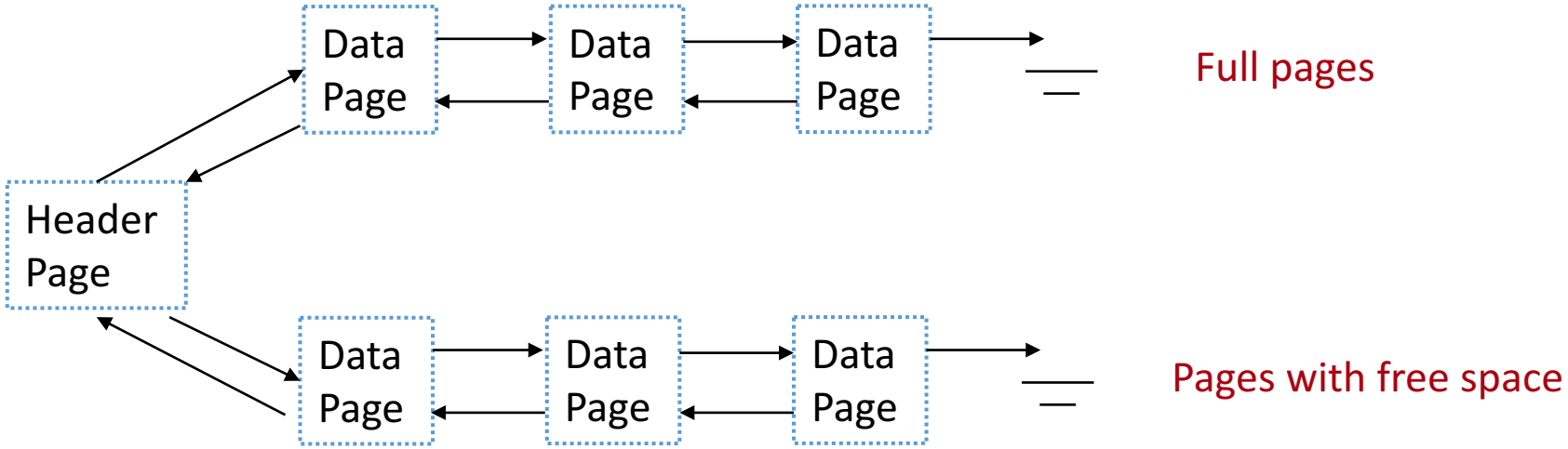
1. Recap: Heap Files (Alles in Ordnung)
2. Why Indexes
3. Index Basics
4. **Indexes in Practice**

1. Recap: Heap Files

File Organization: Unordered (Heap) Files

- Simplest file structure contains records in no particular order.
- As file grows and shrinks, disk pages are allocated and de-allocated.
- To support record level operations, we must:
  - keep track of the *pages* in a file: page id (pid)
  - keep track of *free space* on pages
  - keep track of the *records* on a page: record id (rid)
  - Many alternatives for keeping track of this information
- Operations: create/destroy file, insert/delete record, fetch a record with a specified rid, scan all records

Heap File as a List



- (heap file name, header page id) recorded in a known location
- Each page contains two pointers plus data: Pointer = Page ID (pid)
- Pages in the free space list have "some" free space

Q: What happens with variable length records?

A: All pages are going to have free space, but maybe we will have to go through a lot of them before we find one with enough space.

2. Why Indexes

“If you don’t find it in the index, look very carefully through the entire catalog”

- - Sears, Roebuck and Co., Consumers Guide, 1897

Real Motivation

- Consider the following SQL query:

```
SELECT *
FROM Sales
WHERE Sales.date = “02-11-2016”
```

- For a heap file, we have to scan all the pages of the file to return the correct result

Alternative File Organizations

- We can speed up the query execution by better organizing the data in a file
- There are many alternatives:
  - sorted files
  - indexes
    - B+ tree
    - Hash index
    - Bitmap index

3. Index Basics

Indexes

- An **Index**: speeds up searches for a subset of records, based on values in certain (*search key*) fields
  - any subset of the fields of a relation can be the search key
  - a search key is *not* the same as the primary key

- An index contains a collection of *data entries* (each entry with enough info to locate the records)

An **index** is a data structure that organizes records to optimize retrieval.

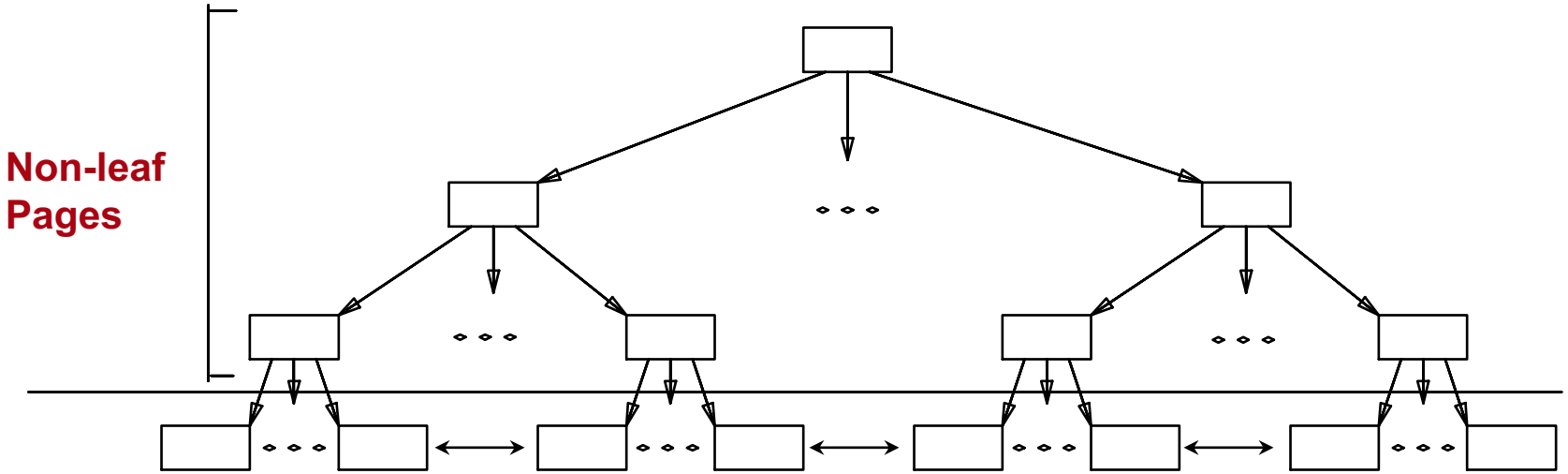
Example: Hash Index

- A hash index is a collection of buckets
  - bucket = primary page plus overflow pages
  - buckets contain data entries

- uses a hash function **h**
  - ***h(r)*** = bucket in which (data entry for) record *r* belongs

- good for equality search
- not so good for range search (use tree indexes instead)

Example: B+ Tree Index



- Leaf pages contain data entries, and are chained (prev & next)
- Non-leaf pages have data entries

Index Data Entries

- The actual data may not be in the same file as the index
- In a data entry with search key **k** we have 3 alternatives of what to store:
  - **Alternative 1:** the record with key value **k**
  - **Alternative 2:** <**k**, rid of record with search key value **k**>
  - **Alternative 3:** <**k**, list of rids of records with search key **k**>
- The choice of alternative for data entries is independent of the indexing technique

Alternatives for Data Entries

Alternative #1:

- index structure is a file organization for records
- at most one index on a given collection of data records (why?)
- if data records are very large, the number of pages containing data entries is high (slower search)

Alternatives for Data Entries

Alternatives #2 and #3:

- Data entries are typically much smaller than data records. So, better than #1 with large data records, especially if search keys are small

- #3 is more compact than #2, but leads to variable sized data entries even if search keys are of fixed length

More on Indexes

- A file can have several indexes
- Index classification:
  - *Primary* vs *secondary*
  - *Clustered* vs *unclustered*

Primary vs Secondary

- If the search key contains the primary key, it is called a primary index
- Any other index is called a secondary index
- If the search key contains a candidate key, it is called a unique index
  - a unique index can return no duplicates

Example

`Sales (sid, product, date, price)`

1. An index on (sid) is a primary and unique index
2. An index on (date) is a secondary, but not unique, index

Clustered Indexes

- If the order of records is the same as, or `close to', the order of data entries, it is a **clustered** index
  - alternative #1 implies clustered
  - in practice, clustered also implies #1
  - a file can be clustered on at most one search key
  - the cost of retrieving data records through the index varies greatly based on whether index is clustered or not

4. Indexes in Practice

Choosing Indexes

- What indexes should we create?
  - which relations should have indexes?
  - what field(s) should be the search key?
  - should we build several or one index?

- For each index, what kind of an index should it be?
  - clustered
  - hash or tree

Choosing Indexes

- Attributes in **`WHERE`** clause are candidates for index keys
  - exact match condition suggests hash index
  - indexes also speed up joins (later in class)
  - range query suggests tree index (B+ tree)

- Multi-attribute search keys should be considered when a **`WHERE`** clause contains several conditions
  - order of attributes is important for range queries
  - such indexes can enable index-only strategies for queries

Choosing Indexes

Composite search keys: search on a combination of fields
(e.g. <date, price>)

- equality query: every field value is equal to a constant value
  - date="02-20-2015" and price =75
- range query: some field value is not a constant
  - date="02-20-2015"
  - date="02-20-2015" and price > 40

Indexes in SQL

```
CREATE INDEX index_name
ON table_name (column_name);
```

- Example of simple search key:

```
CREATE INDEX index1
ON Sales (price);
```

Indexes in SQL

```
CREATE UNIQUE INDEX index2
ON Sales (sid);
```

- A unique index does not allow any duplicate values to be inserted into the table
- It can be used to check integrity constraints (a duplicate value will not be allowed to be inserted)

Indexes in SQL

```
CREATE INDEX index3
ON Sales (date, price);
```

- Indexes with composite search keys are larger and more expensive to update
- They can be used if we have multiple selection conditions in our queries

Summary

- Indexes
  - alternative file organization
- Index classifications:
  - hash vs tree
  - clustered vs unclustered
  - primary vs secondary