CS 133: Databases

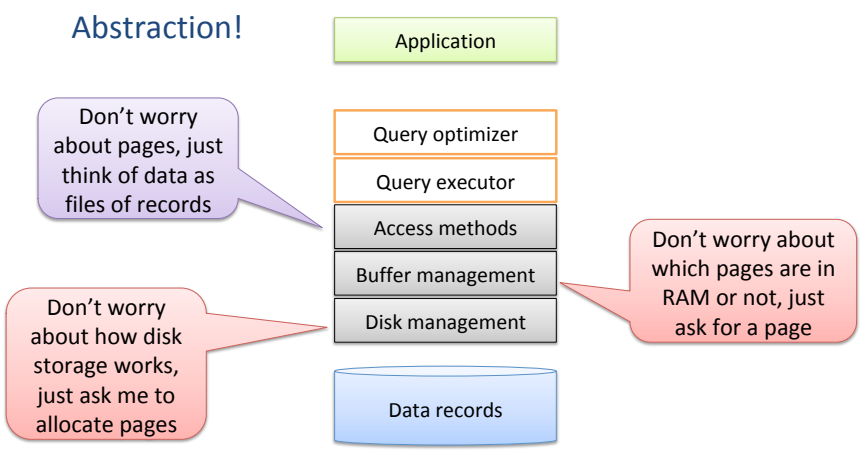
Fall 2019
Lec 03 – 09/10
Files and Records

Prof. Beth Trushkowsky

Goals for Today

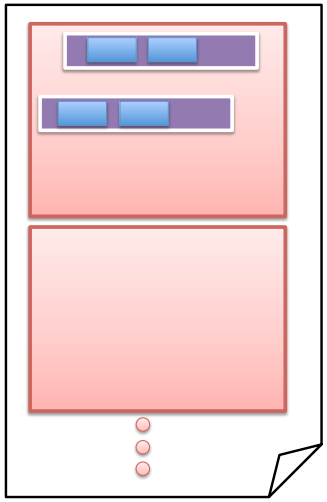
- Reason about the tradeoffs between different ways to organize files, records, and fields on disk
- Calculate the cost of finding records within different file organizations
- Discuss the role of iterators in processing queries in a relational DBMS

Simplified RDBMS Architecture



File Organization

- File is a logical collection of **pages**
- Each page has a set of **records (tuples)**
- Each record has a set of **fields**



Files of Records

- Higher levels of DBMS operate on *records* and *files of records*
- File API must support:
  - insert/delete/modify record
  - fetch a particular record (specified using ***record id***)
  - scan all records (possibly with some conditions on the records to be retrieved)

System Catalogs

- How do we know **which file** to look at for a given relation?
  - **Catalog** stores metadata ("data about data")
- For **each relation**:
  - name, file name, file organization (e.g., Heap file)
  - attribute name and type, for each attribute
  - index name, for each index
  - integrity constraints
- (Plus statistics, authorization, buffer pool size, indexes, …)

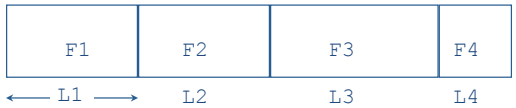
In many DBMSs, catalog itself stored as a relation!

Record Data Types and Sizes

```
CREATE TABLE Courses (
  course_id CHAR(5),
  name VARCHAR(100),
  units INTEGER,
  PRIMARY KEY(course_id)
);
```

Range of data type options varies by DBMS

Record Format: Fixed Length



F*i*: the *i*th field
L*i*: length of the *i*th field in bytes

Info found in system catalog!

```
SELECT major FROM Students;
```

Q1. Suppose `major` is third field, how to get it from the record?

Q2. Implications of `major` being blank (*NULL*) for some records?

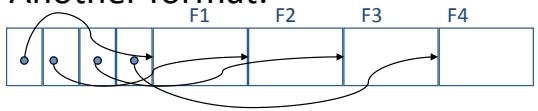
Record Format: Variable Length

- One possible format:

Fields Delimited by Special Symbols

How can we access the third field?

- Another format:



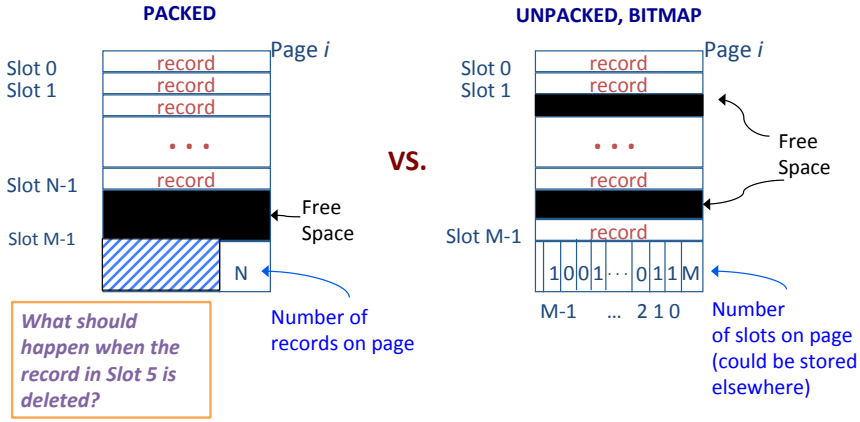
Array of Field Offsets

Second format offers direct access to *i*th field, efficient storage of *nulls* (special *don't know* value); some directory overhead.

Organizing Records on Pages

- ***Important***: ability to locate individual records by some identifier (id)
  - E.g., indexes may need these ids
  - Typically the **record id** is <page id, slot #>

Page Formats: Fixed length Records



Exercise 2: Fitting Records on a Page

a). Each tuple has 2 int and one string field, so 4+4+128 = 136

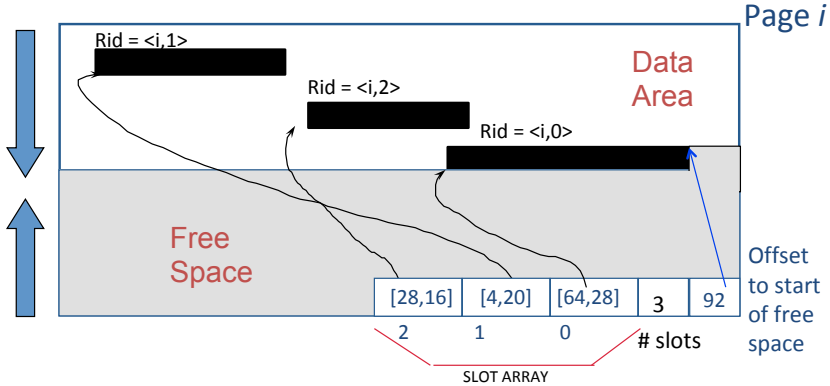
b) # bits for each tuple, including header: 136*8 +1

numTuples = floor((1024 * 8) / (136 * 8 + 1)) = 7

c) headerBytes = ceiling(numTuples / 8) = 1

Check for potential Java rounding issues

Page Format: Variable-length Records



- Slot contains: [offset (from start of page), length]
  - both in bytes
- *Record id = <page id, slot #>*
- Page is full when data space and slot array meet.

Alternative File Organizations

- Different organizations, tradeoffs
- Heap files
  - Unordered *(not the heap data structure)*
  - Easy to maintain!
- Sorted files
  - Great for retrieval in *search key* order
  - Expensive to maintain
- Clustered files (with indexes)
  - Compromise between these extremes

Comparing File Organizations: Cost Analysis

- Average-case analysis; based on several simplistic assumptions
- Ignore CPU costs, for simplicity:
  - **B:** The number of data blocks (pages), no wasted space
- Simply count number of disk block I/Os
  - This ignores gains of pre-fetching and sequential access, so I/O cost is approximate

Good enough to show some overall trends!

Heap Files (Unordered)

- As file grows and shrinks, disk pages are allocated and de-allocated
- For cost analysis, we'll assume:
  - **Insert** always appends to end of file
  - **Delete** just leaves free space in the page
  - Empty pages are not de-allocated
  - **Dirty pages** written back to disk
- Queries such as:

Students(**sid**, name, gpa)

```
SELECT * FROM Students WHERE sid=18;

SELECT * FROM Students WHERE gpa > 3.0;

INSERT INTO Students VALUES(18, 'Beth', 4.0);

DELETE FROM Students WHERE sid=18;
```

Sorted Files

- Sorted files **maintain the sorted property** on update
- Assumptions for analysis
  - Keep pages fully packed, **no gaps**
  - Searches are on the sort key field(s)

Exercise 3(b)

Average Case I/O Counts (B = # disk blocks in file)

	Heap File	Sorted File
Scan all records	B	B
Equality Search (1 match)	0.5 B	$\log_2$ B (on sort key) 0.5 B (otherwise)
Range Search	B	($\log_2$ B) + *selectivity* * B
Insert (1 record)	2	($\log_2$B)+ 2*blocks to move
Delete (1 record)	0.5B+1	Same cost as Insert (if close gap)

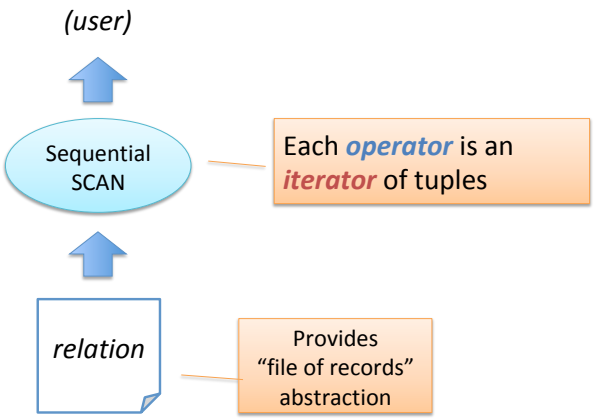
Iterators!

- What are they used for?
- Why are they great?

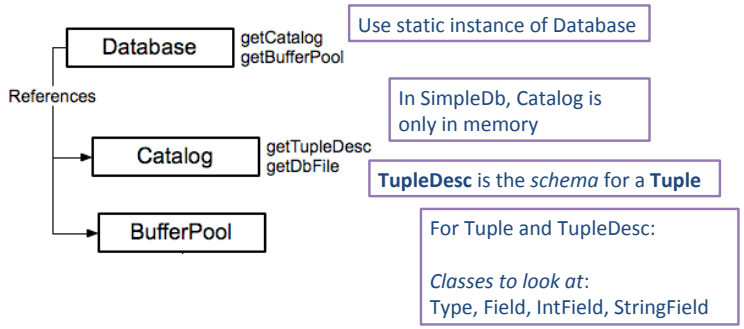
SimpleDb: A Basic DBMS

Goal of Lab 1: Support queries like
`SELECT * FROM relation`

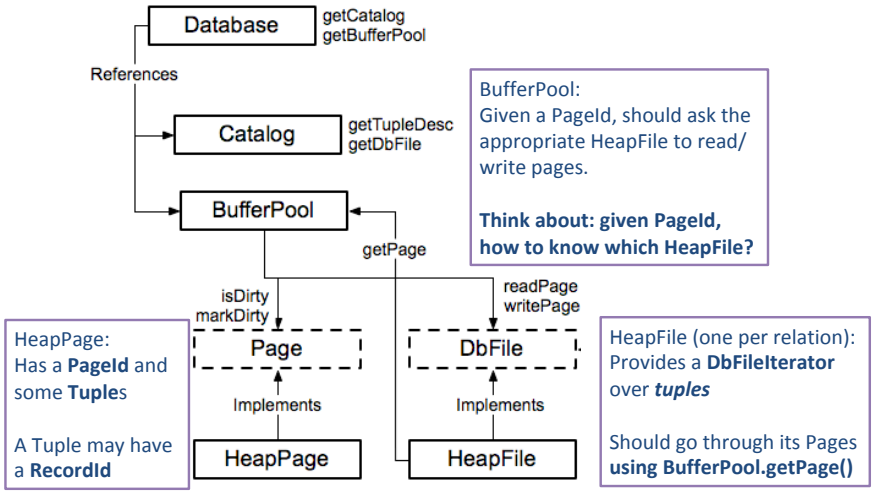
(well, without a nice user interface)



SimbleDb: Module Diagram



SimbleDb: Module Diagram



SimbleDb: Module Diagram