DB internals

First steps

We’ll assume our system has to process SQL.

We’ll assume the data is so large that it can’t be stored in memory.

We’ll assume the data is stored primarily on a hard drive.

Query processing

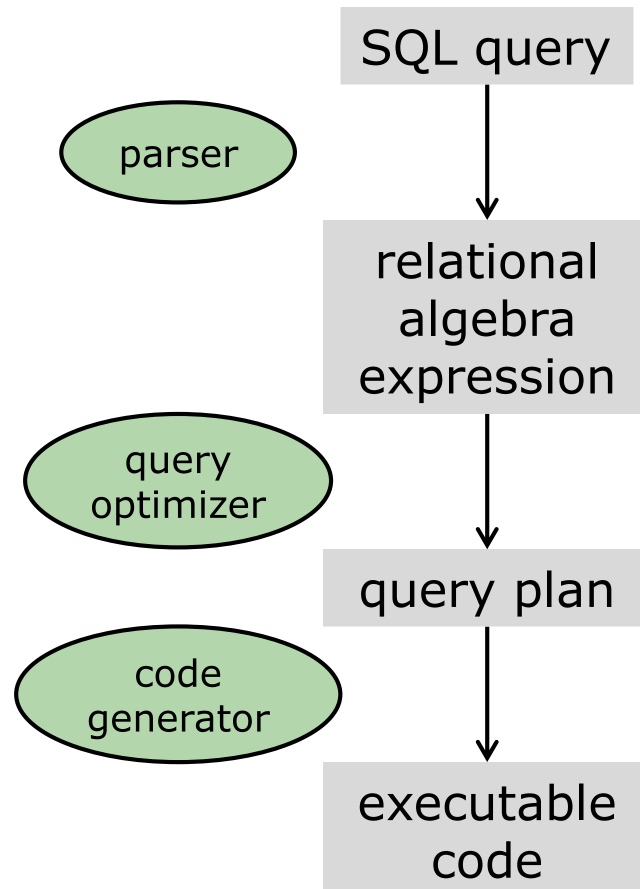
Exercise:

What happens when you run an SQL query?

Query processing

General steps in query processing:

1. parse the SQL
   - check syntax, convert to internal form
2. type check the query
   - are tables, attributes in the database?
3. convert to relational algebra
4. execute the relational algebra



Query optimization

SQL is declarative, so there are many ways to run a query.

- multiple ways to convert query to relational algebra
- multiple ways to evaluate a relational algebra expression

Query optimization

A problem is to choose a “query plan”.

How to determine cost of alternative query plans?

Many relevant factors:

- computational time
- I/O time
- required memory/disk space
- impact on time for DB updates

Estimating the cost of running a query

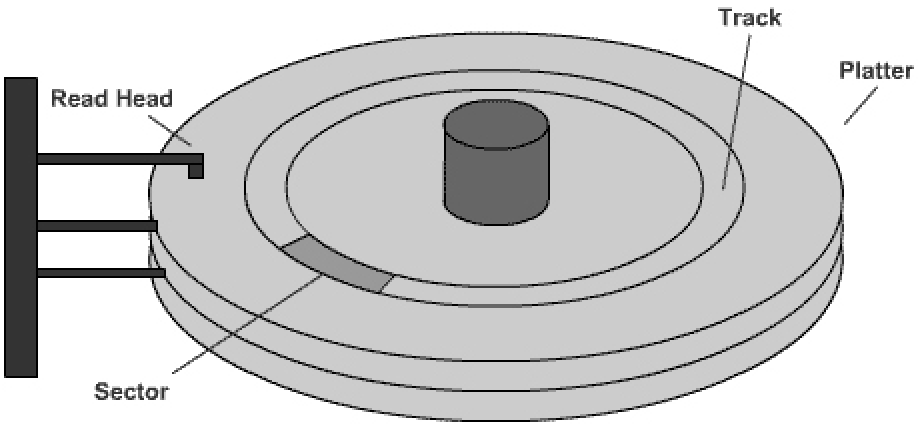
- Exercise:
  - what is the biggest factor affecting the time it takes to run an SQL query (on a large DB)?

Estimating the cost of running a query

- Databases are generally too big to put in memory, so we need to use disk (hard drive)
- Reading from disk is <u>much</u> slower than reading from memory.
- So, in optimizing query evaluation we want to minimize the number of disk accesses

Basic of hard drives

- Data is stored on tracks, sectors
  - 1 track ~ 1000 sectors of 512 bytes each
- Access time: ~ 10 ms
- Transfer time: ~ 80 MB/sec
- Block: unit of transfer between disk and memory
- Typical block size: 4K or 8K bytes



(figure from http://www.911-computer.com/how-to-repair-hard-drive-bad-sectors)

Reducing disk block access

- store related records in the same block
- use write buffers

How to store a database on disk?

- To simplify things, we assume:
  - data for each table in its own file
  - all records of a table are the same size
- A **database** = a collection of files
- A **file** is a sequence of blocks
- A **block** is a sequence of records
- A **record** is a sequence of fields

Example

We'll assume one file for each table.

We can simply store the records in a file, one after another

instructor

record	ID	name	dept_name	salary
0	10101	Srinivasan	Comp. Sci.	65000
1	12121	Wu	Finance	90000
2	15151	Mozart	Music	40000
3	22222	Einstein	Physics	95000
4	32343	El Said	History	60000
5	33456	Gold	Physics	87000
6	45565	Katz	Comp. Sci.	75000
7	58583	Califieri	History	62000
...	76543	Singh	Finance	80000
	76766	Crick	Biology	72000
	83821	Brandt	Comp. Sci.	92000
	98345	Kim	Elec. Eng.	80000

Buffering disk data

- We want to minimize the number of blocks transferred between disk and memory
- So – blocks are kept in memory if possible
- Buffer manager
  - allocates buffer space (in memory)
  - programs that want data call the buffer manager:
    - if block is in memory, manager returns it
    - otherwise, manager gets block from disk

In-Memory Search

Idea: linear search

Let's pretend data is in memory.

select name from student where dept_name = "Comp. Sci.";

Simplest strategy is to search sequentially from top to bottom.

In general we must search all records.

student

ID	name	dept_name	tot_cred
74732	Zhang	Comp. Sci.	102
12345	Shankar	Comp. Sci.	32
50337	Brandt	History	80
23121	Chavez	Finance	110
35432	Avery	Comp. Sci.	44
44553	Peltier	Physics	56
23145	Levy	Physics	46
54321	Williams	Comp. Sci.	54
88232	Sanchez	Music	38
70557	Snow	Physics	0
20346	Brown	Comp. Sci.	58
76653	Aoi	Elec. Eng.	60
12543	Bourikas	Elec. Eng.	98
46524	Tanaka	Biology	120

Special case 1: equality on key

select * from student where ID = 44553;

If search is for equality on primary key attribute, stop when found.

Search half the table in average case.

student

ID	name	dept_name	tot_cred
74732	Zhang	Comp. Sci.	102
12345	Shankar	Comp. Sci.	32
50337	Brandt	History	80
23121	Chavez	Finance	110
35432	Avery	Comp. Sci.	44
44553	Peltier	Physics	56
23145	Levy	Physics	46
54321	Williams	Comp. Sci.	54
88232	Sanchez	Music	38
70557	Snow	Physics	0
20346	Brown	Comp. Sci.	58
76653	Aoi	Elec. Eng.	60
12543	Bourikas	Elec. Eng.	98
46524	Tanaka	Biology	120

Special case 2: ordered field

select name from student where total_cred < 50;

If linear search on ordered field, can also stop early.

Search half the table in average case.

student

ID	name	dept_name	tot_cred
70557	Snow	Physics	0
12345	Shankar	Comp. Sci.	32
55739	Sanchez	Music	38
35432	Avery	Comp. Sci.	44
45678	Levy	Physics	46
54321	Williams	Comp. Sci.	54
44553	Peltier	Physics	56
76543	Brown	Comp. Sci.	58
76653	Aoi	Elec. Eng.	60
19991	Brandt	History	80
98765	Bourikas	Elec. Eng.	98
128	Zhang	Comp. Sci.	102
23121	Chavez	Finance	110
98988	Tanaka	Biology	120

Linear search: pros and cons

Pros:

- ultra-simple
- no special data structures needed
- no overhead when records added/deleted

Cons:

- slow: may need to search entire table

Binary search

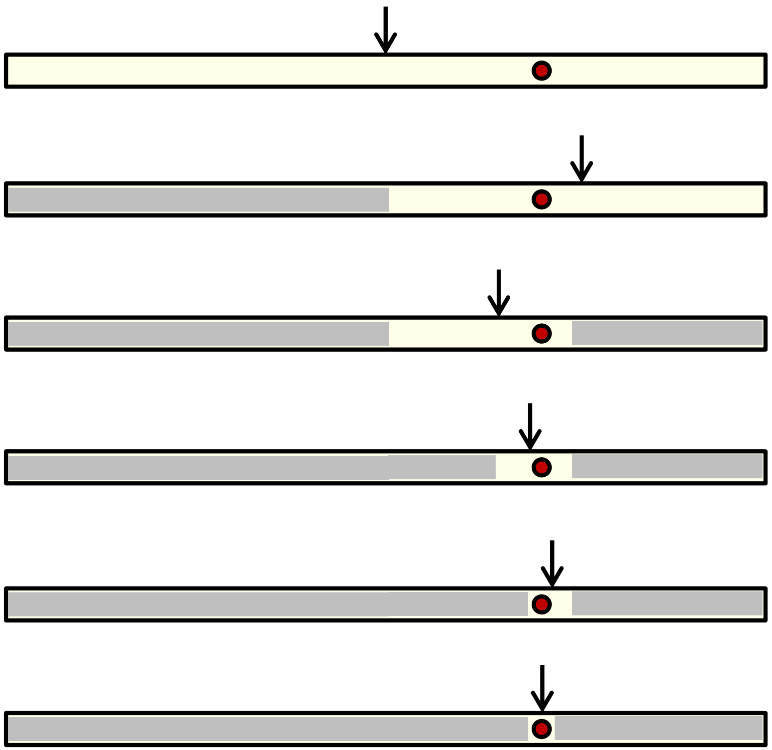
select name from student where ID = 76543;

If field is ordered, can do **binary search**.

student

ID	name	dept_name	tot_cred
128	Zhang	Comp. Sci.	102
12345	Shankar	Comp. Sci.	32
19991	Brandt	History	80
23121	Chavez	Finance	110
35432	Avery	Comp. Sci.	44
44553	Peltier	Physics	56
45678	Levy	Physics	46
54321	Williams	Comp. Sci.	54
55739	Sanchez	Music	38
70557	Snow	Physics	0
76543	Brown	Comp. Sci.	58
76653	Aoi	Elec. Eng.	60
98765	Bourikas	Elec. Eng.	98
98988	Tanaka	Biology	120

Binary search



student

ID	name	dept_name	tot_cred
128	Zhang	Comp. Sci.	102
12345	Shankar	Comp. Sci.	32
19991	Brandt	History	80
23121	Chavez	Finance	110
35432	Avery	Comp. Sci.	44
44553	Peltier	Physics	56
45678	Levy	Physics	46
54321	Williams	Comp. Sci.	54
55739	Sanchez	Music	38
70557	Snow	Physics	0
76543	Brown	Comp. Sci.	58
76653	Aoi	Elec. Eng.	60
98765	Bourikas	Elec. Eng.	98
98988	Tanaka	Biology	120

Binary search: pros and cons

Pros:

- pretty simple
- much faster than linear search

Cons:

- only one field of a table can be sorted
- overhead to keep data sorted when rows are added, deleted, modified

On-Disk Search

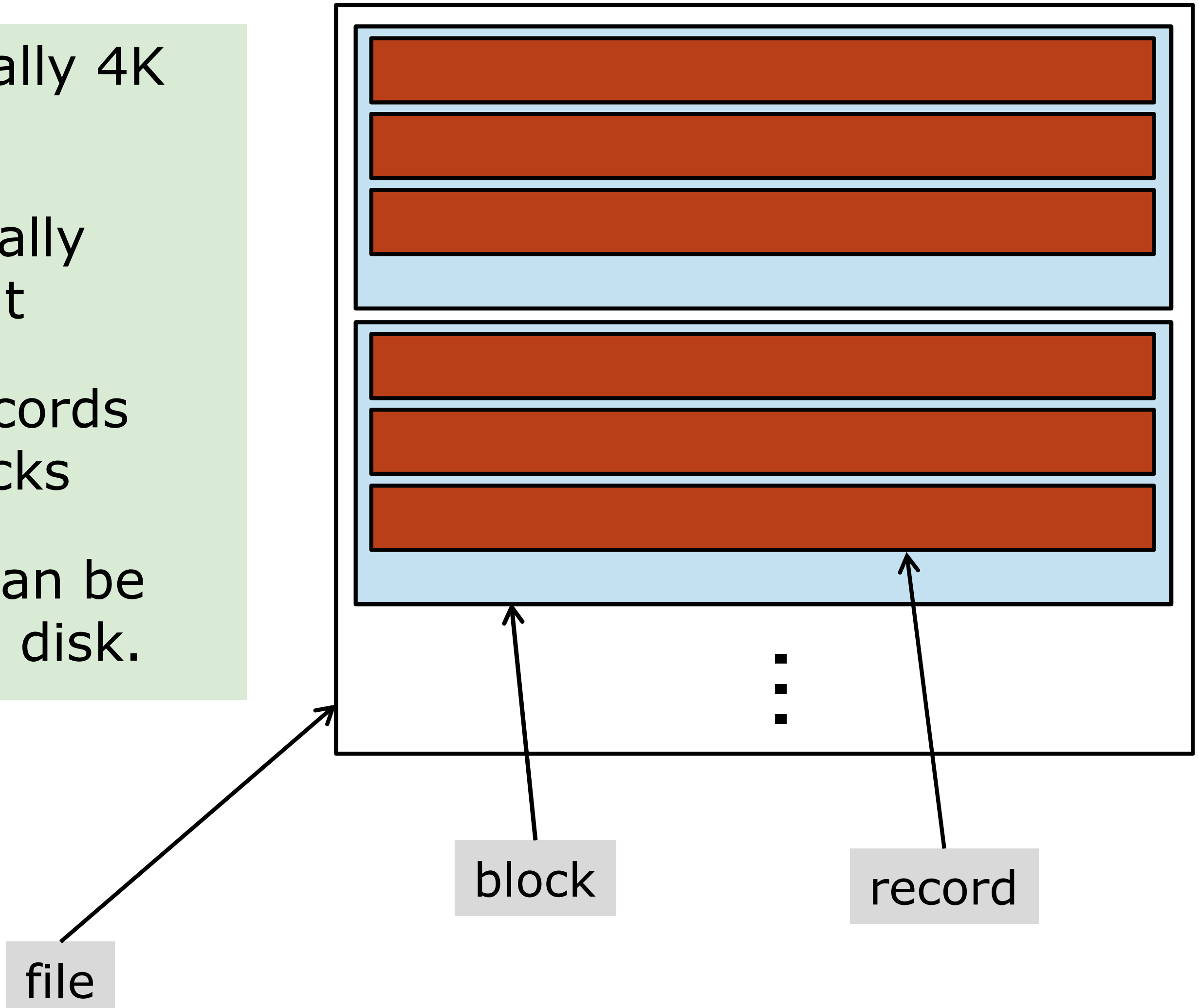
Records and blocks

Blocks are typically 4K or 8K bytes

Records are usually smaller than that

We'll assume records don't bridge blocks

Blocks of a file can be scattered across disk.



Linear search over table on disk

With data on disk, our cost estimate is based on time spent on disk I/O.

Linear search, *assuming blocks are stored contiguously on disk*:

access_time + num_blocks * transfer_time

typical access_time: 4 ms

typical transfer_time: 0.1 ms

typical block size: 4K or 8K bytes

Note: transfer_time is the time to transfer one block.
It can be derived from the disk transfer if you know the block size.

Binary search

With binary search, we will have to “skip around” the disk, even if file blocks are contiguous.

Also, without additional data structures, how do you find the address of a block in the “middle” of some rows?

File organization

How are the records ordered within a file?

heap organization:

- records are in an arbitrary order

sequential organization

- records are ordered according to the value of one attribute
- this attribute is called the “search key”

Sequential files

Ideally, in sequential organization, records are physically ordered according to the search key.

How to cope with insertion and deletion of records?

student

ID	name	dept_name	tot_cred
128	Zhang	Comp. Sci.	102
12345	Shankar	Comp. Sci.	32
19991	Brandt	History	80
23121	Chavez	Finance	110
35432	Avery	Comp. Sci.	44
~~44553~~	~~Peltier~~	~~Physics~~	~~56~~
45678	Levy	Physics	46
54321	Williams	Comp. Sci.	54
55739	Sanchez	Music	38
70557	Snow	Physics	0
76543	Brown	Comp. Sci.	58
76653	Aoi	Elec. Eng.	60
98765	Bourikas	Elec. Eng.	98
98988	Tanaka	Biology	120

Sequential file organization

instructor

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

A pointer in each record points to the next record.

Essentially a linked list.

Every so often the file is reorganized.