CS 133: Databases

Fall 2019
Lec 02 – 09/05

Relational Model & Memory and Buffer Manager

Prof. Beth Trushkowsky

Administrivia

- Problem set 1 out tonight, due Thursday 11:59pm
  – Honor code: can use lectures notes and textbook, can discuss *general ideas* with classmates
  – On Sakai
- Lab 1: "Getting started" due Wednesday
  – On course website
  (labs will also be linked from assignment on Sakai)
  – Nothing to submit yet… eventual submission on Gradescope

Goals for Today

- Reason about the conceptual evaluation of an SQL query
- Understand the storage hierarchy and why disk input/output (I/O) is an important metric for query cost
- See how different policies for managing which data stays in RAM can impact cost of queries

Relational Model

- Users write declarative queries using logical schema
  – May actually interact with application that queries the database
  – Database administrator (DBA) typically creates database
- Given declarative query, DBMS figures out efficient execution strategy

We'll start discussion of "choices" today!

College database

Students

SID	name	login	gpa
45	Alice	alicious	3.4
67	Bob	bobtastic	3.9

Courses

CID	name	credits
CS 121	Software Dev	3
CS 70	Data Structures	3

Courses (***cid***: string, *name:* string, *credits:* integer)

Multi-Relation Queries

Students

SID	name	login	gpa
45	Alice	alicious	3.4
67	Bob	bobtastic	3.9
78	Carl	carl	2.5

Enrolled

SID	CID	grade
45	CS133	A
45	CS121	B
78	CS5	A

SELECT S.name, E.CID
FROM **Students** S, **Enrolled** E
WHERE S.sid=E.sid AND E.grade = "B";

S.name	E.CID
Alice	CS121

Basic Query: Select-From-Where

```
SELECT [DISTINCT] A1, A2, …, An
FROM R1, R2, …, Rn
WHERE condition(s);
```

Relation List.
Relations used in query, implicitly *JOIN*ed.

Target List
Attributes from relation list.

Comparisons. Conjunctive ("AND"), and Disjunctive ("OR")

Also called an SPJ (select-project-join)

Query Semantics

Conceptual query evaluation steps:

1. do FROM clause: *cross-product* of tables
2. do WHERE clause: check conditions, discard tuples that fail
3. do SELECT clause: delete unwanted fields
4. do DISTINCT: eliminate duplicate tuples
   (SQL SELECT defaults to keeping duplicates)

Actually very inefficient in practice!
An optimizer will find more efficient strategies to get the same answer.

SELECT S.name, E.CID
FROM Students S, Enrolled E
WHERE S.sid=E.sid AND E.grade = "B";

(1) FROM: Cross-Product

FROM Students S, Enrolled E

Students

SID	name	login	gpa
45	Alice	alicious	3.4
67	Bob	bobtastic	3.9
78	Carl	carl	2.5

Enrolled

SID	CID	grade
45	CS133	A
45	CS121	B
78	CS5	A

S.SID	S.name	S.login	S.gpa	E.SID	E.CID	E.grade
45	Alice	alicious	3.4	45	CS133	A
67	Bob	bobtastic	3.9	45	CS133	A
78	Carl	carl	2.5	45	CS133	A
45	Alice	alicious	3.4	45	CS121	B
67	Bob	bobtastic	3.9	45	CS121	B
78	Carl	carl	2.5	45	CS121	B
45	Alice	alicious	3.4	78	CS5	A
67	Bob	bobtastic	3.9	78	CS5	A
78	Carl	carl	2.5	78	CS5	A

SELECT S.name, E.CID
FROM Students S, Enrolled E
WHERE S.sid=E.sid AND E.grade = “B”;

(2) WHERE: Discard tuples that fail conditions

WHERE S.sid=E.sid AND E.grade=‘B’

Students X Enrolled

	S.SID	S.name	S.login	S.gpa	E.SID	E.CID	E.grade
✖	45	Alice	alicious	3.4	45	CS133	A
✖	67	Bob	bobtastic	3.9	45	CS133	A
✖	78	Carl	carl	2.5	45	CS133	A
	45	Alice	alicious	3.4	45	CS121	B
✖	67	Bob	bobtastic	3.9	45	CS121	B
✖	78	Carl	carl	2.5	45	CS121	B
✖	45	Alice	alicious	3.4	78	CS5	A
✖	67	Bob	bobtastic	3.9	78	CS5	A
✖	78	Carl	carl	2.5	78	CS5	A

SELECT S.name, E.CID
FROM Students S, Enrolled E
WHERE S.sid=E.sid AND E.grade = “B”;

(3) SELECT: Delete Unwanted Fields

SELECT S.name, E.CID

S.SID	S.name	S.login	S.gpa	E.SID	E.CID	E.grade
✖		✖	✖	✖		✖
45	Alice	alicious	3.4	45	CS121	B

S.name	E.CID
Alice	CS121

Exercise 2: Writing SQL

```
Students (sid, name, login, gpa)
Courses (cid, name)
Enrolled (sid, cid, grade)
```

Write an SQL query that finds the course ***cid*** for ***only the courses that gave at least one A grade***

Aggregation

Enrolled

SID	CID	grade
45	CS133	A
45	CS121	B
67	CS133	A
78	CS5	A

- What does this query produce?

```
SELECT COUNT(*)
FROM Enrolled E;
```

- Built-in Aggregates: COUNT, SUM, AVG, MAX, MIN
- What about the count of enrollments *per* course?

```
SELECT cid, COUNT(*) courseCount
FROM Enrolled E
GROUP BY cid;
```

cid	courseCount
CS133	2
CS121	1
CS5	1

- Enrollments for only “large” classes

```
SELECT COUNT(sid)
FROM Enrolled E
GROUP BY cid
HAVING COUNT(sid) > 50;
```

[Less] Basic Query Anatomy

SELECT [DISTINCT] $A_1, A_2, \ldots, A_n$

FROM $R_1, R_2, \ldots, R_n$

WHERE *condition(s)*

GROUP BY $A_1, A_2, \ldots, A_n$ — *Grouping list.* Attributes from relation list.

HAVING *conditions(s)*; — *Group qualifications.* Conditions on each group.

Query Semantics (cntd)

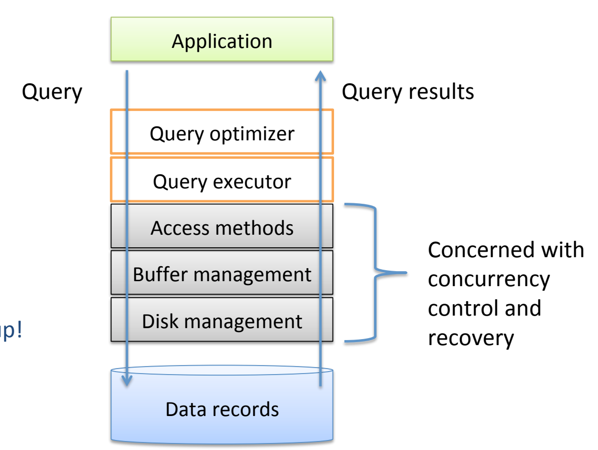
Conceptual query evaluation steps:

1. do **FROM** clause: *cross-product* of tables
2. do **WHERE** clause: check conditions, discard tuples that fail
3. Remove fields not in **SELECT**, **GROUP BY**, or **HAVING** clauses
4. do **GROUP BY**: partition into groups
5. do **HAVING**: delete groups that do not meet conditions

Could do #3 after #5 also

***Result*: one answer tuple per qualifying group**

Simplified RDBMS Architecture



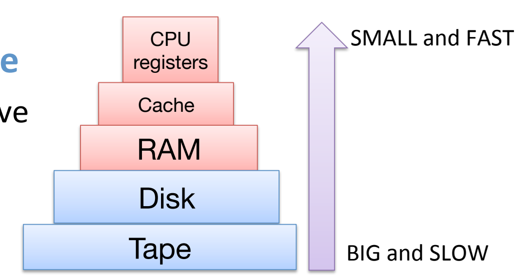
Let's look at the system bottom-up!

Computer Storage

- **Primary storage**
  - E.g., "main memory" a.k.a random-access memory (**RAM**)
  - Typically ***volatile***

Where variable values go when your program is running!

- **Secondary storage**
  - E.g., hard disk drive
  - Non-volatile



Why Not Keep All Data in Memory?

- Costs too much!
  - $100 for 100 GB of RAM or around 2 TB of disk
  - Databases can be in the petabyte (1000 TB) range
- Main memory volatile
  - Want persistence

A Typical Disk

- Moving parts!
  - Platters spin
  - Arms move in/out to position heads with track
  - Tracks under heads make conceptual cylinder

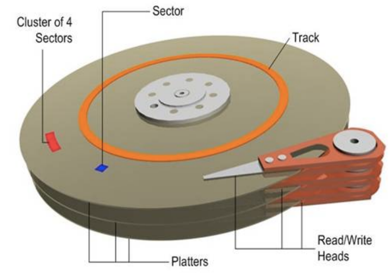


Photo: http://i.technet.microsoft.com/dynimg/IC306536.jpg

- A *block* is a unit of transfer
  - made up of one or more sectors

In main memory, we'll call this chunk of data a ***page***

Disk Access Time

- Time to read/write (an Input/Output or **I/O**) a block
  - Seek time
  - Rotational Delay
  - Transfer time
- **Seek** time and **rotational delay** dominate (stats: wikipedia)
  - Seek time: about 4 to 15msec
  - Rotational: avg 4msec (7200rpm)
  - Transfer rate: < 0.1msec per 8KB block

Reduce I/O cost by reducing seek and rotation

Reading from disk to RAM

Exercise 3: Counting I/Os

- Query: joining relations **Students** and **Enrolled**

```
SELECT S.name, E.CID
FROM Students S, Enrolled E
WHERE S.sid=E.sid;
```

- [Simple] join pseudocode:

```
For each tuple i of outer relation
    For each tuple j of inner relation
        Check if i.sid == j.sid
```

- Relation info
  - Students: 20 pages, 1000 total tuples
  - Enrolled: 50 pages, 6000 total tuples
  - For a given relation, pages on disk sequentially

Exercise: Counting I/Os

- Think of the simple algorithm as a nested for-loop like this:

```
For each page of Outer relation
    Load that page // one I/O
    For each tuple of Outer on that page
        For each page of Inner relation
            Load that page // one I/O
                For each tuple of Inner on that page
                    // do tuple comparison
```

Inner loop executes once for each tuple of *Outer* relation

Exercise: Counting I/Os

- Total I/Os = (# pages in *outer*) + (# tuples in *outer*) * (# pages in *inner*)
  - Students outer: 20 + 1000*50 = 50,020
  - Enrolled outer: 50 + 6000*20 = 120,050
- # Random I/Os = (# pages in *Outer*) + (# tuples in *Outer*)(1)

Reading the first page of *Inner* will be a random I/O each time

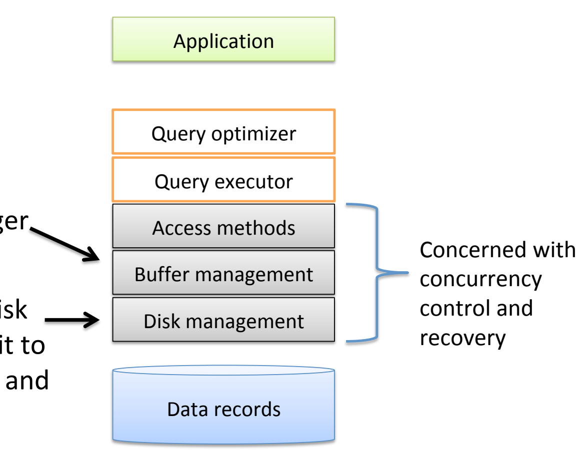
- # Sequential I/Os = (# pages in *Inner* – 1) (# tuples in *Outer*)

Simplified RDBMS Architecture

Next: the buffer manager

Disk Space Manager

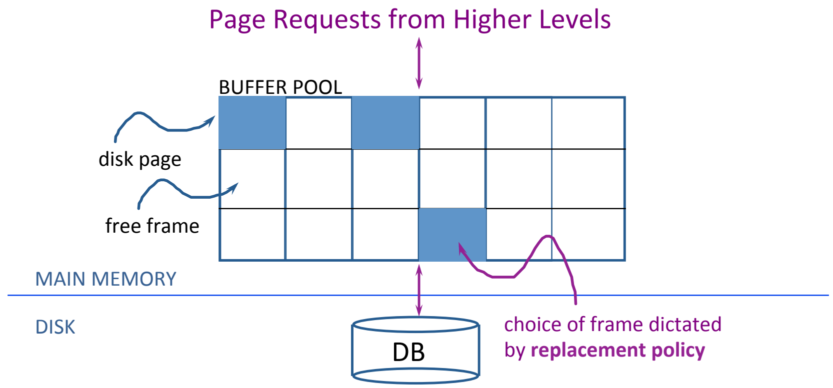
- Manages space on disk
- Higher levels call on it to allocate/de-allocate, and read/write *pages*



The Buffer Manager

- Data must be RAM for DBMS to operate on it
  - Too costly to keep all data in RAM
- Buffer manager
  - Maintain a pool of space in RAM
  - Talks to disk space manager to read/write pages
  - Higher levels do not know what is in RAM or not

Buffer Pool



Important Terms

- Disk page: unit of transfer between disk and memory. Size is DBMS configuration parameter (e.g., 4-32 KB).
- Frame: unit of memory. Typically same size as disk page size.
- Buffer Pool: collection of frames used to temporarily keep data for query processor.

When a Request Comes in…

- If requested page is *in* the buffer pool
  - ***Pin*** the page to mark as in use
- Else, if requested page is *not* in buffer pool
  - If there is an available frame, put the page in that frame
  - Else, select a frame for *replacement* using a **replacement policy**
    (only un-pinned pages are eligible for replacement)
    - If selected frame is **dirty**, write it back to disk
    - Read requested page into the selected frame
    - Pin the page

Pin_count == 0

Buffer Replacement Policy

- When no available frames in buffer pool, need to **evict** one based on a **replacement policy**
  - Choice of policy impacts number of disk I/Os
  - Efficacy depends on ***access pattern*** of pages

What would an optimal policy do?

LRU Policy (Least Recently Used)

- Evict the page that was accessed (pinned) furthest in the past, i.e., **the *least recently used*** of the pages in the pool

Intuition: if a page has not been used in a while, it probably won't again soon

- Example:
  - Buffer pool with 4 frames
  - Assume pages are immediately unpinned after use

Access pattern:	A	B	C	A	D	E	C	B
Frame 1	A							B
Frame 2		B				E		
Frame 3			C					
Frame 4					D			

time →

hits: 2
misses: 6