class 3

Column-Stores Basics

Prof. Manos Athanassouli

https://midas.bu.edu/classes/CS591A1/
Project details are now on-line (more to come)

detailed discussion on Thursday 1/30
Readings for the project


Monkey: Optimal Navigable Key-Value Store by Niv Dayan, Manos Athanassoulis, Stratos Idreos. SIGMOD Conference 2017

More readings (for some research projects)


Small Materialized Aggregates: A Light Weight Index Structure for Data Warehousing by Guido Moerkotte. VLDB 1998

The adaptive radix tree: ARTful indexing for main-memory databases by Viktor Leis, Alfons Kemper, Thomas Neumann. ICDE 2013: 38-49
programming language: C/C++

it gives you control over exactly what is happening
it helps you learn the impact of design decisions

avoid using libraries unless asked to do,
so you can control storage and access patterns
Reviews

8 reviews and the rest single technical question

review (up to one page)
what is the problem & why it is important?
why is it hard & why older approaches are not enough?
what is key idea and why it works?
what is missing and how can we improve this idea?
does the paper supports its claims?
possible next steps of the work presented in the paper?

single technical question

to make sure the heart of the paper is clearly understood
Presentations

for every class, **one or two students will be responsible for presenting** the paper (discussing all main points of a long review – see next slide)

during the presentation **anyone can ask questions** (including me!) and each question is **addressed to all** (including me!)

the presenting student(s) will **prepare slides and questions**
what to do now?

A) read the syllabus and the website
B) register to piazza
C) register to gradescope/blackboard
D) register for the presentation (week 2)
E) start submitting paper reviews (week 3)
F) go over the project (more details on the way)
G) start working on the mid-semester report (week 3)

survival guide

class website: https://midas.bu.edu/classes/CS591A1/
piazza website: http://piazza.com/bu/spring2020/cs591a1/
presentation registration: https://tinyurl.com/S2020-CS591-presentations
gradescope entry-code: 9568G3
office hours: Manos (Tu 2-3pm/F 1-2pm)
TA office hours: see piazza
material: papers available from BU network
how can I prepare?

1) Read background research material

2) Start going over the papers
Database Design Abstraction Levels

- Logical Design
- Physical Design
- System Design
Data can be messy!

clean  $\rightarrow$ schema  $\rightarrow$ ...

?
Data can be messy!

clean ➔ schema ➔ load ➔ ...

?
Data can be messy!

clean → schema → load → tune
Data can be messy!

clean $\rightarrow$ schema $\rightarrow$ load $\rightarrow$ tune

experts and DBAs

query

any user!
Database Design Abstraction Levels

- Logical Design
- Physical Design
- System Design
Logical design

What is our data? How to model them?


Relational!

A collection of **tables**, each being a collection of **rows and columns**

*schema: describes the columns of each table*
Logical design

What is our data? How to model them?

- graph data
- time-series data

Relational:

A collection of **tables**, each being a collection of **rows and columns**

[schema: describes the columns of each table]
Logical Schema of “University” Database

**Students**

- `sid`: string,
- `name`: string,
- `login`: string,
- `year_birth`: integer,
- `gpa`: real

**Courses**

- `cid`: string,
- `cname`: string,
- `credits`: integer

**Enrolled**

- `sid`: string,
- `cid`: string,
- `grade`: string
Relational Model and SQL

**Relations**

- **Students**
  - `sid`: string, `name`: string, `login`: string, `year_birth`: integer, `gpa`: real

- **Courses**
  - `cid`: string, `cname`: string, `credits`: integer

- **Enrolled**
  - `sid`: string, `cid`: string, `grade`: string
Relational Model and SQL

**Students**

- **sid**: string,  
  - **name**: string,  
  - **login**: string,  
  - **year_birth**: integer,  
  - **gpa**: real

**Courses**

- **cid**: string,  
  - **cname**: string,  
  - **credits**: integer

**Enrolled**

- **sid**: string,  
  - **cid**: string,  
  - **grade**: string

---

*how to create the table students?*

```sql
create table students (sid:char(10), name:char(40), login:char(8), age:integer, ...)
```

*how to add a new student?*

```sql
insert into students (U1398217312, John Doe, john19, 19, ...)
```

*bring me the names of all students with GPA > 3.5*

```sql
select name from students where GPA > 3.5
```
Relational Model and SQL

student
(sid1, name1, login1, year1, gpa1)
(sid2, name2, login2, year2, gpa2)
(sid3, name3, login3, year3, gpa3)
(sid4, name4, login4, year4, gpa4)
(sid5, name5, login5, year5, gpa5)
(sid6, name6, login6, year6, gpa6)
(sid7, name7, login7, year7, gpa7)
(sid8, name8, login8, year8, gpa8)
(sid9, name9, login9, year9, gpa9)

insert into student (sid1, name1, login1, year1, gpa1)

cardinality: 9
Relational Model and SQL

**student**
(sid1, name1, login1, year1, gpa1)
(sid2, name2, login2, year2, gpa2)
(sid3, name3, login3, year3, gpa3)
(sid4, name4, login4, year4, gpa4)
(sid5, name5, login5, year5, gpa5)
(sid6, name6, login6, year6, gpa6)
(sid7, name7, login7, year7, gpa7)
(sid8, name8, login8, year8, gpa8)
(sid9, name9, login9, year9, gpa9)

*insert into* student (sid1, name1, login1, year1, gpa1)

Cardinality: 9

What if a student does not have their login yet?
Relational Model and SQL

**student**
(sid1, name1, login1, year1, gpa1)
(sid2, name2, login2, year2, gpa2)
(sid3, name3, login3, year3, gpa3)
(sid4, name4, login4, year4, gpa4)
(sid5, name5, login5, year5, gpa5)
(sid6, name6, login6, year6, gpa6)
(sid7, name7, login7, year7, gpa7)
(sid8, name8, login8, year8, gpa8)
(sid9, name9, **NULL**, year9, gpa9)

**insert into** student (sid1, name1, login1, year1, gpa1)

**cardinality:** 9

what if a student does not have their login yet? **NULL values** do not exist
Relational Model and SQL

Students

- sid: string
- name: string
- login: string
- year_birth: integer
- gpa: real

Courses

- cid: string
- cname: string
- credits: integer

Enrolled

- sid: string
- cid: string
- grade: string

how to show all enrollments in CS591A1?
Relational Model and SQL

**Students**

- `sid`: string, `name`: string, `login`: string, `year_birth`: integer, `gpa`: real

**Courses**

- `cid`: string, `cname`: string, `credits`: integer

**Enrolled**

- `sid`: string, `cid`: string, `grade`: string

Using foreign keys we can join information of all three tables:

```
select student.name
from students, courses, enrolled
where course.cname="CS591A1"
and course.cid=enrolled.cid
and student.sid=enrolled.sid
```
Database Design Abstraction Levels

- Logical Design
- Physical Design
- System Design
Physical Design

File Organization

- heap files
- sorted files
- clustered files
- more ...

Indexes

- should I build?
- on which attributes/tables?
- what index structure?
- B-Tree
- Tries
- Hash
- Bitmap
- Zonemaps
Data systems are declarative!

ask **what** you want

data system

system decides **how** to store & access

design decisions, physical design, indexing, tuning knobs

research to automate!

adaptivity

autotuning
Database Design Abstraction Levels

- Logical Design
- Physical Design
- System Design
select max(B) from R where A>5 and C<10
select max(B) from R where A>5 and C<10
**memory wall**

- **cache miss**: looking for something that is not in the cache
- **memory miss**: looking for something that is not in memory

Diagram:
- CPU
- on-chip cache
- on-board cache
- main memory
- flash storage
- disks
- flash

Graph:
- Performance vs. Time
  - Old times!
  - CPU
  - DRAM
Jim Gray, IBM, Tandem, Microsoft, DEC
“The Fourth Paradigm” is based on his vision
ACM Turing Award 1998
ACM SIGMOD Edgar F. Codd Innovations award 1993
data movement & page-based access

Data movement through all necessary levels:
- CPU
  - on-chip cache
  - on-board cache
  - main memory
  - flash storage
  - disks
  - flash

Need to read only X.

Also read unnecessary data.
access granularity

file system and DBMS “pages”
data storage

Student (sid: string, name: string, login: string, year_birth: integer, gpa: real)

how to physically place data?
slotted page

header

row1

row2

row3

free space
slotted page

#rows, row offsets, free space offsets,
#fixed length attributes, #var length attributes

row1

row2

row3

free space
querying over slotted pages

```sql
select A,B,C,D from R
```

```sql
select A from R
```

each page contains **entire** rows (all their columns)

rows are **contiguous**

(with possible free space at the end)
querying over slotted pages

**schema: R (A,B,C,D)**

- select A,B,C,D from R
- select A from R
- select (A+B) from R

each page contains columns!
querying over slotted pages

schema: \( R \) \((A,B,C,D)\)

- \( \text{select } A, B, C, D \text{ from } R \)
- \( \text{select } A \text{ from } R \)
- \( \text{select } (A+B) \text{ from } R \)

each page contains \textit{columns} or \textit{groups of columns}!

what if I had both queries?
not clear!
other hybrids?
what if only inserts?
column-stores history line

- 1985: first complete column-store model
- 2000: first complete column-store system
- 2012+: expanding on hybrid layouts
- 2001: first idea for hybrid layouts

Time periods:
- 60s
- 70s
- 80s
- 90s
- 00s
- 10s
- 20s

Data scales:
- Rows
- Rows
- Rows
- Rows
- Rows
- Rows
- Rows*
the way we physical store data dictates what are the possible efficient access methods
query evaluation
select max(B) from R where A>5 and C<10

one row at a time
select max(B) from R where A>5 and C<10

tuple reconstruction/early materialization

one row at a time

late materialization

column at a time
select max(B) from R where A>5 and C<10

int* input=A;
int* output; /*needs allocation*/
for (i=0; i<num_tuples; i++,input++)
  if (*input>5)
  {
    *output=i;
    output++;
  }
select max(B) from R where A > 5 and C < 10

what is the benefit?
sequential access patterns
read only useful data
**easy to code**: working over fixed width and dense columns

**scan**

```plaintext
for (i=0, j=0; i<size; i++)
    if (column[i] qualifies)
        res[j++]=i;
```

- no complex checks
- no function calls
- no aux metadata
- easy to prefetch
- as few ifs as possible

**fetch**

```plaintext
for (i=0, j=0; i<fetch_size; i++)
    intermediate_result[j++]=column[ids[i]];
```
**select** \( \text{max}(B) \) **from** \( R \) **where** \( A > 5 \) **and** \( C < 10 \)

- alternatives query plans
  - scan \( A \) & \( C \) in parallel and merge
  - start from \( C \) (why?)
  - use bit vectors (why?)
select max(B) from R where A>5 and C<10

whole column?
row at a time
column at a time
block/vector at a time
```sql
select max(B) from R where A>5 and C<10
```
why column-stores are here now?

late materialization – no need to reconstruct tuples
read only useful data
minimize data movement across the memory hierarchy
but it required a complete re-write

why not before?
legacy technology to catch up
more important: analytical workloads (as opposed to only OLTP)
new hardware: larger memories & memory wall
class 3

Column-Stores Basics

Prof. Manos Athanassoulis

https://midas.bu.edu/classes/CS591A1/