Database Systems I, CSCI-GA.2433-001
New York University, Fall 2019

instructor: Dennis Shasha
shasha@cs.nyu.edu
212-998-3086
Courant Institute
New York University
251 Mercer Street
NY, NY 10012 USA
Office Hours: On Mondays by appointment before class in Warren Weaver lobby

December 7, 2019

1 Goals

The course seeks to achieve three goals:

1. To help you understand the design of data intensive systems given an underlying database management systems (i.e. entity-relationship conceptual design, normalization theory, query languages).
2. To introduce you to internals (indexes, query processing).
3. To introduce you to some advanced topics (analytical processing, data cleaning and preparation, introduction to concurrency control and recovery and to distribution).

Most of the course notes we use were written by Professor Zvi Kedem, though I've modified a little (and at times skip some slides). They are all here on the website. Here is the order of presentation:

- 01_Goals_Of_The_Course.pptx

- 05_SQL_As_Data_Manipulation_Language.pptx
- 06_SQL_As_Data_Definition_And_Control_Language.pptx
- 02_Modeling_Enterprise_With_ER_Diagrams.pptx
- 03_From_ER_Diagrams_To_Relational_Databases.pptx
- 07_Logical_Design_With_Normalization.pptx
- 04_Relational_Algebra_With_SQL_Equivalents.pptx
- 08_Physical_Design_And_Query_Execution_Concepts.pptx
- 11_Online_Analytical_Processing.pptx
- 12_Advanced_Concepts.pptx NoSQL databases, distribution
- If time, some introduction to data cleaning, transaction processing and recovery from the advanced database course.

2 Mechanics

YOU MUST BE ENROLLED IN THIS CLASS TO SIT IN ON THE LECTURES.

2.1 Texts and Notes

There are no textbooks, but you will look up manuals (notably for mySQL and for data structure libraries) on the web.

2.2 Prerequisites

Fundamental Algorithms I is a co-requisite.

2.3 Course Requirements

three problem sets (40%), project (60%). There are no exams in this course.

LATE HOMEWORKS OR PROJECTS WILL NOT BE ACCEPTED without a note from your physician or from your employer. (We will discuss the solutions on the day you hand in the assignment. That's why I don't want any late homeworks. As for the project, this is a question of fairness.)

On the other hand, collaboration on the problem sets IS allowed. You may work together with one other student and sign both of your names to a single submitted homework. Both of you will receive the grade that the homework merits. So, you may work alone or in a team of two, but no team larger than two. Please choose your partner carefully.

2.4 How to Set Up MySQL

You will also need to set up a database using mysql by following the instructions in homework 1.

Here are some commands:

```
show databases;
show tables;
create table friends(name1 varchar(20), name2 varchar(20));
insert into friends values("bob", "alice");
select * from friends;
```

Now suppose that you have a file foo.csv with vertical bar delimiters on your local machine (in my case in /dbcourse1.d/foo.csv)

```
carol|ted
susan|paul
tyler|cloe
```

You can then import that data into friends as follows:

```
LOAD DATA LOCAL INFILE '~/dbcourse1.d/foo.csv' INTO TABLE friends FIELDS TERMINATED
```

2.5 Project: A miniature relational database with order

This project is due on Thursday Dec 5, 2019 at 4:30 PM.

Given ordered tables (array-tables) whose rows consist of strings (string constants are bracketed by single quotes) and integers, you are to write a program which will

- Perform the basic operations of relational algebra: selection, projection, join, group by, and count, sum and avg aggregates. The comparators for select and join will be $=$ $<$, $>$, $!=$, $\geq$, $\leq$
- Because the array-tables are potentially ordered, you can sort an array-table by one or more columns, and running moving sums and average aggregates on a column of an array-table.
- Import a vertical bar delimited file into an array-table (in the same order), export from an array-table to a file preserving its order, and assign the result of a query to an array-table.
- Each operation will be on a single line. Each time you execute a line, you should print the time it took to execute.
- You will support in memory B-trees and hash structures. You are welcome to take those implementations from wherever you can find them, but you must say where.
- Your program should be written in python or java. You will hand in clean and well structured source code in which each function has a header that says (i) what the function deos, (ii) what its inputs are and what they mean (iii) what the outputs are and mean (iv) any side effects to globals.
- You must ensure that your software runs on the Courant Institute (cims) machine crunchy5.cims.nyu.edu
- You may NOT use any relational algebra or SQL library or system (e.g. no SQLite, no mySQL, no other relational database system, no Pandas). Stick pretty much to the standard stuff (e.g. in Python: numpy, core language features, string manipulation, random number generators, and data structure support for in memory B-trees and hash structures). You may not use anyone else's code (other than for the

B-tree or Hash structure implementation). Doing so will constitute plagiarism.

Your program should take operations from standard input. We will run your programs on test cases of our choosing. For ease of parsing there will be one operation per line. Comments begin with // and go to the end of the line. For example,

```
R := inputfromfile(sales1) // import vertical bar delimited foo, first line
    // has column headers.
    // Suppose they are saleid|itemid|customerid|storeid|time|qty|pricerange
    // In general there can be more or fewer columns than this.
R1 := select(R, (time > 50) or (qty < 30))
   // select * from R where time > 50 or qty < 30
R2 := project(R1, saleid, qty, pricerange)  // select saleid, qty, pricerange
   // from R1
R3 := avg(R1, qty) // select avg(qty) from R1
R4 := sumgroup(R1, time, qty) // select sum(time), qty from R1 group by qty
R5 := sumgroup(R1, qty, time, pricerange) // select sum(qty), time,
  // pricerange from R1 group by time, pricerange
R6 := avggroup(R1, qty, pricerange) // select avg(qty), pricerange
 // from R1 group by by pricerange
S := inputfromfile(sales2) // suppose column headers are
    // saleid|I|C|S|T|Q|P
T := join(R, S, R.customerid = S.C) // select * from R, S
   // where R.customerid = S.C
T1 := join(R1, S, (R1.qty > S.Q) and (R1.saleid = S.saleid)) // select * from R1, S w
T2 := sort(T1, S_C) // sort T1 by S_C
T2prime := sort(T1, R1_time, S_C) // sort T1 by R1_time, S_C (in that order)
T3 := movavg(T2prime, R1_qty, 3) // perform the three item moving average of T2prime
   // on column R_qty. This will be as long as R_qty with the three way
   // moving average of 4 8 9 7 being 4 6 7 8
T4 := movsum(T2prime, R1_qty, 5) // perform the five item moving sum of T2prime
   // on column R_qty
Q1 := select(R, qty = 5) // select * from R where qty=5
Btree(R, qty) // create an index on R based on column qty
   // Equality selections and joins on R should use the index.
   // All indexes will be on one column (both Btree and Hash)
Q2 := select(R, qty = 5) // this should use the index
```

```
Q3 := select(R, itemid = 7) // select * from R where itemid = 7
Hash(R,itemid)
Q4 := select(R, itemid = 7) // this should use the hash index
Q5 := concat(Q4, Q2) // concatenate the two tables (must have the same schema)
   // Duplicate rows may result (though not with this example).
outputtofile(Q5, Q5) // This should output the table Q5 into a file
   // with the same name and  with vertical bar separators
outputtofile(T, T) // This should output the table T
```

Our tests may operate on different files with different column headers. Our queries may use different paramter values (e.g. 14 way moving average). Our joins may be on different fields.

Some constraints to make your life easier:

- There will be no syntax errors in our tests. However, white space may vary in the operations, e.g. select(R, itemid = 7), select(R, itemid = 7), select(R,itemid=7), and select(R ,itemid = 7) all should be interpreted in the same way.
- The only aggregates are count, sum, and avg and the corresponding countgroup, sumgroup, and avggroup.
- The only moving aggregates are movsum and movavg. There is no group by for moving sums and averages.
- All data is in main memory.
- Within select there may be several conditions but only all ands or all ors.
- Within join there may be several conditions but only all ands.
- A join condition can be
  table.attribute [arithop constant] relop table.attribute [artithop constant]
- A select condition can be
  attribute [arithop constant] relop constant
  or
  constant relop attribute [arithop constant]

- If there are multiple conditions, each condition will be surrounded by parentheses.

- `relop is =, !=, >, >=, <, <=`
  `arithop is +, -, *, /`

- `If R is (A,B,C) and S is (B,C,D,E) then`
  `the result of the join will be`
  `R_A, R_B, R_C, S_B, S_C, S_D, S_E.`

- All numbers should be interpreted as floating point. So 5/2 = 2.5

- project syntax: project(TABLENAME, listofatts) listofatts is just a comma-separated list of attributes. e.g. R2 := project(R1, saleid, qty, pricerange)

- sum and avg take a table and a single attribute e.g. R3 := avg(R1, qty) // select avg(qty) from R1

- sumgroup, avggroup, countgroup take a table, a single attribute and a list of attributes and perform the aggregate on the single attribute grouped by the other attributes, All these primitives should be assumed to execute on a multiset (i.e. duplicates are NOT removed). e.g. R5 := sumgroup(R1, qty, time, pricerange) // select sum(qty), time, // pricerange from R1 group by time, pricerange R5 has a single column R5(sumgroup).

- sort takes a table and one or more attributes and sorts by those attributes in order, e.g. T2prime := sort(T1, R1_time, S_C) // sort T1 by R1_time, S_C (in that order) Column names of T2prime are the same as for T1.

- movavg and movsum take a table, an, attribute and a constant k and does the k-way moving aggregate of that attribute, e.g. T3 := movavg(T2prime, R1_qty, 3) // perform the three item moving average of T2prime Resulting table has a field with multiple values, e.g. T3(movavg).

- concat takes two tables with the same schema and concatenates each colum in order e.g. Q5 := concat(Q4, Q2) // concatenate the two tables (must have the same schema) Q5 has the same column names as Q4 and Q2.

- count(TABNAME) // count the number of rows in the table even if duplicate rows If this is assigned to a table R, then R has a single column name, viz. R(count).

Here is an example of the first few lines of the sales1 file:

```
saleid|itemid|customerid|storeid|time|qty|pricerange
45|133|2|63|49|23|outrageous
658|75|2|89|46|43|outrageous
149|103|2|23|67|2|cheap
398|82|2|41|3|27|outrageous
147|81|2|4|92|11|outrageous
778|75|160|72|67|17|supercheap
829|112|2|70|63|43|supercheap
101|105|2|9|74|28|expensive
940|62|2|90|67|39|outrageous
864|119|12|38|67|49|outrageous
288|46|2|95|67|26|outrageous
875|83|59|56|59|20|outrageous
783|86|180|29|67|46|outrageous
289|16|2|95|92|2|cheap
```

Full example files can be found here:

```
http://cs.nyu.edu/cs/faculty/shasha/papers/sales1
http://cs.nyu.edu/cs/faculty/shasha/papers/sales2
```

You will hand in your code using Reprozip inside a Docker Virtual Machine so that it is reproducible across platforms.